Chapter . The Type Traits Introspection Library 1.4

Edward Diener

Table of Contents

Introduction

Welcome to the Type Traits Introspection library version 1.4.

The Type Traits Introspection library, or TTI for short, is a library of macros generating metafunctions, and a set of parallel nullary type metafunctions, which provide the ability to introspect by name the elements of a type at compile time.

The name of the library is chosen because the library offers compile time functionality on a type, similar to the Boost Type Traits library, and because the functionality the library offers is the ability to introspect a type about the existence of a specific element.

I use the word "introspect" in a very broad sense here. Normally language introspection means initially asking for information to be returned by name, which can then further be used to introspect for more specific information. In the TTI library one must always supply the name, and use the functionality for the correct type of inner element to find out if the particular named entity exists. You may prefer the term "query" instead of "introspection" to denote what this library does, but I use terminology based on the word "introspect" throughout this documentation.

The functionality of the library may be summed up as:

- Provide the means to introspect a type at compile time using a set of macros. Each macro takes the name of the type's element and generates a metafunction which can be subsequently invoked to determine whether or not the element exists within the type. These metafunctions will be called "macro metafunctions" in the documentation.
- Provide corresponding metafunctions which can operate on almost all of the macro metafunctions generated. These secondary metafunctions provide the same set of introspection as the macro metafunctions but allow for an easier to use syntax, where the types being passed are in the form of nullary metafunctions themselves. They always work with individual types when specifying function syntax rather than a composite type. These metafunctions are called 'nullary type metafunctions' in the documentation.
- Provide a set of macros for all of the macro metafunctions which create metafunction classes, so the metafunctions generated by the macro metafunctions can be passed as metadata. These can be used by the nullary type metafunctions, and may find other uses for the template metaprogrammer.

The library is dependent on Boost PP, Boost MPL, Boost Type Traits, and Boost Function Types. The library is also dependent on the `variadic_macro_data` library currently in the sandbox if the variadic macros in the library are used.

The library is a header only library.

Since the dependencies of the library are all header only libraries, there is no need to build anything in order to use the TTI library.

Header Files

There are two separate general header files in the library, divided depending on whether or not the library functionality supporting variadic macros is to be used, which bring in all the rest of the specific header files.

1. The main header, which does not require using the library support for variadic macros, is `tti.hpp`. This can be used for the vast majority of functionality in the library.
2. The secondary header, which uses a small subset of the library functionality, providing an alternate use of particular macros with variadic macro support, is `tti_vm.hpp`.

There are also separate specific header files for each of the elements to be introspected by the library. This allows for finer-grained inclusion of nested elements to be introspected. These header files are:

Table 1. TTI Header Files

Inner Element	File
Type	`type.hpp`
Template	`template.hpp`
Template with params	`template_params.hpp`
Template with params - variadic macros	`vm_template_params.hpp`
Member data	`mem_data.hpp`
Member function	`mem_fun.hpp`
Member function - composite syntax	`comp_mem_fun.hpp`
Static member data	`static_mem_data.hpp`
Static member function	`static_mem_fun.hpp`
Static member function - composite syntax	`comp_static_mem_fun.hpp`
Member type	`mem_type.hpp`

If the general header file `tti_vm.hpp` or the specific header file `vm_template_params.hpp`, are used the library is also dependent on the `variadic_macro_data` library currently in the sandbox. If this header is not used there is no need to download the `variadic_macro_data` library and use it in any way.

Why the TTI Library ?

In the Boost Type Traits library there is compile time functionality for querying information about a C++ type. This information is very useful during template metaprogramming and forms the basis, along with the constructs of the Boost MPL library, and some other compile time libraries, for much of the template metaprogramming in Boost.

One area which is mostly missing in the Type Traits library is the ability to determine what C++ inner elements are part of a type, where the inner element may be a nested type, function or data member, static function or static data member, or class template.

There has been some of this functionality in Boost, both in already existing libraries and in libraries on which others have worked but which were never submitted for acceptance into Boost. An example with an existing Boost library is Boost MPL, where there is functionality, in the form of macros and metafunctions, to determine whether an enclosing type has a particular nested type or nested class template. An example with a library which was never submitted to Boost is the Concept Traits Library from which much of the functionality of this library, related to type traits, was taken and improved upon.

It may also be possible that some other Boost libraries, highly dependent on advanced template metaprogramming techniques, also have internal functionality to introspect a type's elements at compile time. But to the best of my knowledge this sort of functionality has never been incorporated in a single Boost library. This library is an attempt to do so, and to bring a recognizable set of interfaces to compile-time type introspection to Boost so that other metaprogramming libraries can use them for their own needs.

Terminology

The term "enclosing type" refers to the type which is being introspected. This type is always a class, struct, or union.

The term "inner xxx", where xxx is some element of the enclosing type, refers to either a type, template, function, or data within the enclosing type. The term "inner element" also refers to any one of these entities in general.

I use the term "nested type" to refer to a type within another type. I use the term "member function" or "member data" to refer to non-static functions or data that are part of the enclosing type. I use the term "static member function" or "static member data" to refer to static functions or data that are part of the enclosing type. I use the term "nested class template" to refer to a class template nested within the enclosing type.

Other terminology may be just as valid for the notion of C++ language elements within a type, but I have chosen these terms to be consistent.

General Functionality

The elements about which a template metaprogrammer might be interested in finding out at compile time about a type are:

- Does it have a nested type with a particular name ?
- Does it have a nested type with a particular name which is a typedef for a particular type ?
- Does it have a nested class template with a particular name ?
- Does it have a nested class template with a particular name and a particular signature ?
- Does it have a member function with a particular name and a particular signature ?
- Does it have a member data with a particular name and of a particular type ?
- Does it have a static member function with a particular name and a particular signature ?
- Does it have a static member data with a particular name and of a particular type ?

These are the compile-time questions which the TTI library answers.

All of the questions above attempt to find an answer about an inner element with a particular name. In order to do this using template metaprogramming, macros are used so that the name of the inner element can be passed to the macro. The macro will then generate an appropriate metafunction, which the template metaprogrammer can then use to introspect the information that is needed. The name itself of the inner element is always passed to the macro as a macro parameter, but other macro parameters may also be needed in some cases.

All of the macros start with the prefix `BOOST_TTI_`, create their metafunctions in a namespace called 'boost::tti', and come in two forms:

1. In the simplest macro form the 'name' of the inner element is used directly to generate the name of the metafunction as well as serving as the 'name' to introspect. To generate the name of the metafunction the 'name' is appended to the name of the macro, with the `BOOST_TTI_` prefix removed, a final underscore added, and the macro parameter 'name' in lower case. As an example, for the macro `BOOST_TTI_HAS_TYPE(MyType)` the name of the metafunction is `boost::tti::has_type_MyType` and it will look for an inner type called 'MyType'.

2. In the more complicated macro form, which I call the complex form, the macro starts with `BOOST_TTI_TRAIT_` and a 'trait' name is passed as the first parameter, with the 'name' of the inner element as the second parameter. The 'trait' name serves solely to completely name the metafunction in the `boost::tti` namespace. As an example, for the macro `BOOST_TTI_TRAIT_HAS_TYPE(MyTrait,MyType)` the name of the metafunction is `boost::tti::MyTrait` and it will look for an inner type called `MyType`. Every macro has a corresponding complex form.

Once either of these two macro forms are used for a particular type of inner element, the corresponding macro metafunction has the exact same functionality.

Important

When introspecting a particular inner element any given macro metafunction generated can be reused with any combination of template parameters which involve the same type of inner element. Furthermore once a macro metafunction is generated, attempting to generate another macro metafunction of the same name will create ODR violations since two C++ constructs with the same name/type in the same namespace will have been created. This latter possibility has much less chance of occurence if you use the simple form of each macro and just reuse the macro metafunction. You can even do this if you are introspecting for two entities of the same name in different enclosing types, or in the same enclosing type but with different signatures, as with overloaded member functions.

In the succeeding documentation all macro metafunctions will be referred by their simple form name, but remember that the complex form name can always alternatively be used.

Macro Metafunction Name Generation

The library also provides a set of macros which can be used to automatically generate the names of the macro metafunctions so that the end-user does not have to remember the naming scheme given above.

For each of the macros using the simple form there are two corresponding macros based on the simple form macro name, each of which takes the name of the inner element as the single macro parameter.

1. The first has the macro metafunction name with `_GEN_BASE` appended to it, and outputs the name of the metafunction without a namespace.

2. The second has the macro metafunction name with `_GEN` appended to it, and outputs the full name of the macro metafunction within the Boost TTI namespace.

These simple form name generation macros are automatically included when the appropriate macro metafunction header file is included.

For the macros using the complicated form, there are two fixed name macros, each of which takes the 'trait' name as the single macro parameter.

1. `BOOST_TTI_TRAIT_GEN_BASE(trait)` simply outputs 'trait' without a namespace.

2. `BOOST_TTI_TRAIT_GEN(trait)` outputs 'trait' within the BOOST TTI namespace.

Both of these last macros are trivial, but are provided nonetheless to parallel the way that the simple form generation macros output their macro metafunction names.

Finally there is a convenience macro which simply outputs the name of the full TTI namespace.

1. `BOOST_TTI_NAMESPACE` simply outputs the full TTI namespace.

These complicated form macros are included by all TTI header files.

Macro Metafunctions

The TTI library uses macros to create metafunctions, in the boost::tti namespace, for introspecting an inner element by name. Each macro for a particular type of inner element has two forms, the simple one where the first macro parameter designating the 'name' of the inner element is used to create the name of the metafunction, and the complex one where the first macro parameter, called 'trait', designates the name of the metafunction and the second macro parameter designates the 'name' to be introspected. Other than that difference, the two forms of the macro produce the exact same results.

To use these metafunctions you can include the main general header file 'tti.hpp', unless otherwise noted. Alternatively you can include a specific header file as given in the table below.

A table of these macros is given, based on the inner element whose existence the metaprogrammer is introspecting. A more detailed explanation can be found in the reference section, and examples of usage can be found in the "Using the Macro Metafunctions"

section. In the Template column only the name generated by the simple form of the template is given since the name generated by the complex form is always boost::tti::trait where 'trait' is the first parameter to the corresponding complex form macro. All of the introspecting metafunctions in the table below return a boolean constant called 'value', which specifies whether or not the inner element exists.

Table 2. TTI Macro Metafunctions

Inner Element	Macro	Template	Specific Header File
Type	BOOST_TTI_HAS_TYPE(name)	boost::tti::has_type_'name' class T = enclosing type	type.hpp
Type with check	BOOST_TTI_HAS_TYPE(name)	boost::tti::has_type_'name' class T = enclosing type class U = type to check against	type.hpp
Class Template	BOOST_TTI_HAS_TEMPLATE(name)	boost::tti::has_template_'name' class T = enclosing type All of the template parameters must be 'class' (or 'typename') parameters	template.hpp
Class Template with params	BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS(name,ppSeq[a])	boost::tti::has_template_check_params_'name' class T = enclosing type	template_params.hpp
Class Template with params using variadic macros[b]	BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS(name,...[c])	boost::tti::has_template_check_params_'name' class T = enclosing type	vm_template_params.hpp
Member data	BOOST_TTI_HAS_MEMBER_DATA(name)	boost::tti::has_member_data_'name' class T = enclosing type class R = data type	mem_data.hpp
Member function as individual types	BOOST_TTI_HAS_MEMBER_FUNCTION(name)	boost::tti::has_member_function_'name' class T = enclosing type class R = return type class FS = (optional) function parameter types as a Boost MPL forward sequence. If there are no function parameters this does not have to be specified. Defaults to boost::mpl::vector<>. class TAG = (optional) Boost function_types tag type. Defaults to boost::function_types::null_tag.	mem_fun.hpp

Inner Element	Macro	Template	Specific Header File
Member function as a composite type	BOOST_TTI_HAS_COMP_MEMBER_FUNCTION(name)	boost::tti::has_comp_member_function_'name' class T = pointer to member function The form for T is 'ReturnType (Class::*)(Zero or more comma-separated parameter types)'	comp_mem_fun.hpp
Static member data	BOOST_TTI_HAS_STATIC_MEMBER_DATA(name)	boost::tti::has_static_member_data_'name' class T = enclosing type class Type = data type	static_mem_data.hpp
Static member function as individual types	BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION(name)	boost::tti::has_static_member_function_'name' class T = enclosing type class R = return type class FS = (optional) function parameter types as a Boost MPL forward sequence. If there are no function parameters this does not have to be specified. Defaults to boost::mpl::vector<>. class TAG = (optional) Boost function_types tag type. Defaults to boost::function_types::null_tag.	static_mem_fun.hpp
Static member function as a composite type	BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION(name)	boost::tti::has_comp_static_member_function_'name' class T = enclosing type class Type = function type The form for Type is 'ReturnType (Zero or more comma-separated parameter types)'	comp_static_mem_fun.hpp

[a] A Boost PP data sequence with each comma separated portion of the template parameters as its own sequence element.

[b] General header file is tti_vm.hpp.

[c] The template parameters as variadic data.

There is one other macro which creates a metafunction which does not introspect for the existence of an inner element type, but is very useful nonetheless. Instead the macro metafunction created returns the nested type if it exists, else it returns an unspecified type.

Table 3. TTI Nested Type Macro Metafunction

Inner Element	Macro	Template	Specific Header File
Type	BOOST_TTI_MEMBER_TYPE(name)	boost::tti::member_type_'name' class T = enclosing type returns = the type of 'name' if it exists, else an unspecified type, as a typedef 'type'.	mem_type.hpp

Along with this macro metafunction we have another metafunction which, when passed a 'type', which can be any type but which will generally be used with the type returned by invoking the metafunction generated by BOOST_TTI_MEMBER_TYPE, will tell use whether the type exists or not as a boolean constant.

Table 4. TTI Nested Type Macro Metafunction Existence

Inner Element	Macro	Template	Specific Header File
Type	None	boost::tti::valid_member_type class T = a type returns = true if the type exists, false if it does not. 'Existence' is determined by whether the type does not equal an unspecified type.	mem_type.hpp

The usefulness of the BOOST_TTI_MEMBER_TYPE macro metafunction will be shown in the next topic when I explain the problem of specifying nested types and how TTI solves it.

Nested Types

The problem

The goal of the TTI library is never to produce a compiler error by just using the functionality in the library, whether it is invoking its function-like macros or instantiating the macro metafunctions created by them, and whether the inner element exists or not. In this sense The TTI library macros for introspecting an enclosing type for an inner element work very well. But there is one exception to this general case. That exception is the crux of the discussion regarding nested types which follows.

The metafunctions generated by the TTI macros all work with types, whether in specifying an enclosing type or in specifying the type of some inner element, which may also involve types in the signature of that element, such as a parameter or return type of a function. The C++ notation for a nested type, given an enclosing type 'T' and an inner type 'InnerType', is 'T::InnerType'. If either the enclosing type 'T' does not exist, or the inner type 'InnerType' does not exist within 'T', the expression 'T::InnerType' will give a compiler error if we attempt to use it in our template instantiation of one of TTI's macro metafunctions.

We want to be able to introspect for the existence of inner elements to an enclosing type without producing compiler errors. Of course if we absolutely know what types we have and that a nested type exists, and these declarations are within our scope, we can always use an expression like T::InnerType without error. But this is often not the case when doing template programming since the type being passed to us at compile-time in a class or function template is chosen at instantiation time and is created by the user of a template.

One solution to this is afforded by the library itself. Given an enclosing type 'T' which we know must exist, either because it is a top-level type we know about or it is passed to us in some template as a 'class T' or 'typename T', and given an inner type named 'InnerType' whose existence we would like ascertain, we can use a `BOOST_TTI_HAS_TYPE(InnerType)` macro and it's related `boost::tti::has_type_InnerType` metafunction to determine if the nested type 'InnerType' exists. This solution is perfectly valid and, with Boost MPL's selection metafunctions, we can do compile-time selection to generate the correct template code.

However this does not scale that well syntactically if we need to drill down further from a top-level enclosing type to a deeply nested type, or even to look for some deeply nested type's inner elements. We are going to be generating a great deal of `boost::mpl::if_` and/or `boost::mpl::eval_if` type selection statements to get to some final condition where we know we can generate the compile-time code which we want.

The solution

The TTI library offers a better solution in the form of constructs which work with nested types without producing a compiler error if the nested type does not exist, but still are able to do the introspecting for inner elements that our TTI macro metafunctions do.

We have already seen one of those constructs, the macro `BOOST_TTI_MEMBER_TYPE`, which generates a metafunction based on the name of an inner type. But instead of telling us whether that inner type exists it instead returns a typedef 'type' which is that inner type if it exists, else it is an unspecified type if it does not. In this way we have created a metafunction, very similar in functionality to boost::mpl::identity, but which still returns some unspecified marker 'type' if our nested type is invalid.

We can use the functionality of `BOOST_TTI_MEMBER_TYPE` to construct nested types for our other macro metafunctions, without having to use the T::InnerType syntax and produce a compiler error if no such type actually exists within our scope. We can even do this in deeply nested contexts by stringing together, so to speak, a series of these macro metafunction results.

As an example, given a type T, let us create a metafunction where there is a nested type FindType whose enclosing type is eventually T, as represented by the following structure:

```
struct T
  {
  struct AType
    {
    struct BType
      {
      struct CType
        {
        struct FindType
          {
          };
        }
      };
    };
  };
```

In our TTI code we first create a series of member type macros for each of our nested types:

```
BOOST_TTI_MEMBER_TYPE(FindType)
BOOST_TTI_MEMBER_TYPE(AType)
BOOST_TTI_MEMBER_TYPE(BType)
BOOST_TTI_MEMBER_TYPE(CType)
```

Next we can create a typedef to reflect a nested type called FindType which has the relationship as specified above by instantiating our macro metafunctions.

```
typedef typename
boost::tti::member_type_FindType
  <
  typename boost::tti::member_type_CType
    <
    typename boost::tti::member_type_BType
      <
      typename boost::tti::member_type_AType
        <
        T
        >::type
      >::type
    >::type
  >::type MyFindType;
```

We can use the above typedef to pass the type as FindType to one of our macro metafunctions. FindType may not actually exist but we will not generate a compiler error when we use it.

As one example, let's ask whether FindType has a static member data called MyData of type 'int'. We add:

```
BOOST_TTI_HAS_STATIC_MEMBER_DATA(MyData)
```

Next we create our metafunction:

```
boost::tti::has_static_member_data_MyData
  <
  MyFindType,
  int
  >
```

and use this in our metaprogramming code. Our metafunction now tells us whether the nested type FindType has a static member data called MyData of type 'int', even if FindType does not actually exist as we have specified it as a type. If we had tried to do this using normal C++ nested type notation our metafunction code above would be:

```
boost::tti::has_static_member_data_MyData
  <
  typename T::AType::BType::CType::FindType,
  int
  >
```

But this fails with a compiler error if there is no such nested type, and that is exactly what we do not want in our compile-time metaprogramming code.

We can also directly find out whether the deeply nested type 'FindType' actually exists in a similar manner. Our metafunction would be:

```
BOOST_TTI_HAS_TYPE(FindType)

boost::tti::has_type_FindType
  <
  typename
  boost::tti::member_type_CType
    <
    typename
    boost::tti::member_type_BType
      <
      typename
      boost::tti::member_type_AType
        <
        T
        >::type
      >::type
    >::type
  >
```

Because this duplicates much of our code for the 'MyFindType' typedef to create our nested type, we can instead, and much more easily, pass our type 'MyFindType', since we already have it in the form of a type, to another metafunction called 'boost::tti::valid_member_type', which returns a boolean constant which is 'true' if our nested exists or 'false' if it does not.

Using this functionality with our 'MyFindType' type above we could create the nullary metafunction:

```
boost::tti::valid_member_type
  <
  MyFindType
  >
```

directly instead of replicating the same functionality with our 'boost::tti::has_type_FindType' metafunction.

The using of `BOOST_TTI_MEMBER_TYPE` to create a nested type which may or may not exist, and which can subsequently be used with our macro metafunctions whenever a nested type is required, without producing a compiler error when the type does not actually exist, is the main reason we have separate but similar functionality among our macro metafunctions to determine whether a member data, a member function, or a static member function exists within an enclosing type.

In the more general case, when using `BOOST_TTI_HAS_MEMBER` and `BOOST_TTI_HAS_STATIC_MEMBER`, the signature for the member function and the static member function is a composite type. This makes for a syntactical notation which is easy to specify, but because of that composite type notation we can not use the nested type functionality in `BOOST_TTI_MEMBER_TYPE` very easily. But when we use `BOOST_TTI_HAS_MEMBER_FUNCTION` and `BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION` the composite types in our signatures are broken down into their individual types so that using `BOOST_TTI_MEMBER_TYPE`, if necessary, for any one of the individual types is easy.

A more elegant solution

Although using `BOOST_TTI_MEMBER_TYPE` represents a good solution to creating a nested type without the possible compile-time error of the T::InnerType syntax, reaching in to specify all those ::type expressions, along with their repeated 'typename', does get syntactically tedious.

Because of this the TTI library offers a parallel set of metafunctions to the macro metafunctions where the 'types' specified are themselves nullary metafunctions. This parallel set of metafunctions, using nullary metafunctions to specify individual types, rather than the actual types themselves, are called 'nullary type metafunctions'. In this group there is also a nullary metafunction paralleling our `BOOST_TTI_MEMBER_TYPE` macro metafunction, and therefore a further construct making the specifying of nested types easy and error-free to use.

This group of nullary type metafunctions will be fully explained later.

Using the Macro Metafunctions

Using the macro metafunctions can be illustrated by first creating some hypothetical user-defined type with corresponding nested types and other inner elements. With this type we can illustrate the use of the macro metafunctions. This is just meant to serve as a model for what a type T might entail from within a class or function template.

```
// An enclosing type

struct AType
  {

  // Type

  typedef int AnIntType; // as a typedef

  struct BType // as a nested type
    {
    struct CType
      {
      };
    };

  // Template

  template <class> struct AMemberTemplate { };
  template <class,class,int,class,template <class> class InnerTemplate,class,long> struct ManyPara↵
meters { };
  template <class,class,int,short,class,template <class,int> class InnerTemplate,class> struct More↵
Parameters { };

  // Data

  BType IntBT;

  // Function

  int IntFunction(short) { return 0; }

  // Static Data

  static short DSMember;

  // Static Function

  static int SIntFunction(long,double) { return 2; }

  };
```

I will be using the type above just to illustrate the sort of metaprogramming questions we can ask of some type T which is passed to the template programmer in a class template. Here is what the class template might look like:

```
#include <boost/tti/tti.hpp>

template<class T>
struct OurTemplateClass
  {

  // compile-time template code regarding T

  };
```

Now let us create and invoke the macro metafunctions for each of our inner element types, to see if type T above corresponds to our hypothetical type above. Imagine this being within 'OurTemplateClass' above. In the examples below the same macro is invoked just once to avoid ODR violations.

Type

Does T have a nested type called 'AnIntType' ?

```
BOOST_TTI_HAS_TYPE(AnIntType)

boost::tti::has_type_AnIntType
  <
  T
  >
```

Does T have a nested type called 'BType' ?

```
BOOST_TTI_HAS_TYPE(BType)

boost::tti::has_type_BType
  <
  T
  >
```

Type checking the typedef

Does T have a nested typedef called 'AnIntType' whose type is an 'int' ?

```
boost::tti::has_type_AnIntType
  <
  T,
  int
  >
```

Template

Does T have a nested class template called 'AMemberTemplate' whose template parameters are all types ('class' or 'typename') ?

```
BOOST_TTI_HAS_TEMPLATE(AMemberTemplate)

boost::tti::has_template_AMemberTemplate
  <
  T
  >
```

Template with params

Does T have a nested class template called 'MoreParameters' whose template parameters are specified exactly ?

```
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS(MoreParameters,(class)(class)(int)(short)(class)(tem↵
plate <class)(int> class InnerTemplate)(class))

boost::tti::has_template_check_params_MoreParameters
  <
  T
  >
```

Template with params using variadic macros

[note Include the `tti_vm.hpp` general header file, or the `vm_template_params.hpp` specific header file, when using this macro.]

Does T have a nested class template called 'ManyParameters' whose template parameters are specified exactly ?

```
BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS(ManyParameters,class,class,int,class,tem↵
plate <class> class InnerTemplate,class,long)

boost::tti::has_template_check_params_ManyParameters
  <
  T
  >
```

Member data

Does T have a member data called 'IntBT' whose type is 'AType::BType' ?

```
BOOST_TTI_HAS_MEMBER_DATA(IntBT)

boost::tti::has_member_data_IntBT
  <
  T,
  AType::BType
  >
```

Member function with individual types

Does T have a member function called 'IntFunction' whose type is 'int (short)' ?

```
BOOST_TTI_HAS_MEMBER_FUNCTION(IntFunction)

boost::tti::has_member_function_IntFunction
  <
  T,
  int,
  boost::mpl::vector<short>
  >
```

Member function with composite type

Does T have a member function called 'IntFunction' whose type is 'int (short)' ?

```
BOOST_TTI_HAS_COMP_MEMBER_FUNCTION(IntFunction)

boost::tti::has_comp_member_function_IntFunction
  <
  int (T::*)(short)
  >
```

Static member data

Does T have a static member data called 'DSMember' whose type is 'short' ?

```
BOOST_TTI_HAS_STATIC_MEMBER_DATA(DSMember)

boost::tti::has_static_member_data_DSMember
  <
  T,
  short
  >
```

Static member function with individual types

Does T have a static member function called 'SIntFunction' whose type is 'int (long,double)' ?

```
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION(SIntFunction)

boost::tti::has_static_member_function_SIntFunction
  <
  T,
  int,
  boost::mpl::vector<long,double>
  >
```

Static member function with composite type

Does T have a static member function called 'SIntFunction' whose type is 'int (long,double)' ?

```
BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION(SIntFunction)

boost::tti::has_comp_static_member_function_SIntFunction
  <
  T,
  int (long,double)
  >
```

Member type

Create a nested type T::BType::CType without creating a compiler error if T does not have the nested type BType::CType ?

```
BOOST_TTI_MEMBER_TYPE(BType)
BOOST_TTI_MEMBER_TYPE(CType)

typename
boost::tti::member_type_CType
  <
  typename
  boost::tti::member_type_BType
    <
    T
    >::type
  >::type
```

Member type existence

Does a nested type T::BType::CType, created without creating a compiler error if T does not have the nested type BType::CType, actually exist ?

```
BOOST_TTI_MEMBER_TYPE(BType)
BOOST_TTI_MEMBER_TYPE(CType)

typedef typename
boost::tti::member_type_CType
  <
  typename
  boost::tti::member_type_BType
    <
    T
    >::type
  >::type
AType;

boost::tti::valid_member_type
  <
  AType
  >
```

Macro Metafunctions as Metadata

As specified in the Boost MPL library, there are two ways to pass metafunctions as metadata, and both ways fall under the Boost MPL terminology of 'lambda expressions':

- As a metafunction class
- As a placeholder expression

Using a placeholder expression is the easiest way and does not require the programmer to create a metafunction class for passing the metadata. The syntax for this is fairly simple. The syntax for passing a macro metafunction becomes `macrometafunction<_>` etc. depending on how many parameters are bring passed. Thus for two parameters we would have `macrometafunction<_,_>` etc., with another placeholder (_) added for each subsequent parameter.

However using a placeholder expression may not be the fastest way when considering compile-time speed. Because of this the TTI library provides a set of macros for each of the macro metafunctions which generate a corresponding metafunction class. We will call this set of macros the 'metafunction class macros'.

These macros take exactly the same macro parameters as their corresponding macro metafunctions. They generate a corresponding metafunction class rather than a metafunction itself, which allows us to pass our macro metafunctions as metadata, just as using placeholder expressions allows us to do.

For these metafunction class macros there is a simple form and a complex form name just as there is for the macro metafunctions. For each macro metafunction, the name for the corresponding metafunction class macro is the macro metafunction name with the sequence `MTFC_` (for "MeTaFunction Class") following the `BOOST_TTI_` (or `BOOST_TTI_VM_`) portion of the macro metafunction name. The subsequent metafunction class name, using the simple form, is the same name as the corresponding macro metafunction name but with `mtfc_` prepended to the name.

As with the complex form name of the macro metafunctions, the complex form name of the corresponding metafunction class macros is completely determined by the first 'trait' parameter.

In the following table I do not specify the macro parameters as they are exactly the same as those for their corresponding macro metafunction.

Table 5. TTI Metafunction Classes

Macro Metafunction	Metafunction Class Macro	Metafunction Class Name
BOOST_TTI_HAS_TYPE	BOOST_TTI_MTFC_HAS_TYPE	boost::tti::mtfc_has_type_'name'
BOOST_TTI_TRAIT_HAS_TYPE	BOOST_TTI_MTFC_TRAIT_HAS_TYPE	boost::tti::'trait'
BOOST_TTI_HAS_TEMPLATE	BOOST_TTI_MTFC_HAS_TEMPLATE	boost::tti::mtfc_has_template_'name'
BOOST_TTI_TRAIT_HAS_TEMPLATE	BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE	boost::tti::'trait'
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS	BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS	boost::tti::mtfc_has_template_check_params_'name'
BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS	BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS	boost::tti::'trait'
BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS	BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS	boost::tti::mtfc_has_template_check_params_'name'
BOOST_TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS	BOOST_TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS	boost::tti::'trait'
BOOST_TTI_HAS_MEMBER_DATA	BOOST_TTI_MTFC_HAS_MEMBER_DATA	boost::tti::mtfc_has_member_data_'name'
BOOST_TTI_TRAIT_HAS_MEMBER_DATA	BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_DATA	boost::tti::'trait'
BOOST_TTI_HAS_MEMBER_FUNCTION	BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION	boost::tti::mtfc_has_member_function_'name'
BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION	BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION	boost::tti::'trait'
BOOST_TTI_HAS_COMP_MEMBER_FUNCTION	BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION	boost::tti::mtfc_has_comp_member_function_'name'
BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION	BOOST_TTI_MTFC_TRAIT_HAS_COMP_MEMBER_FUNCTION	boost::tti::'trait'
BOOST_TTI_HAS_STATIC_MEMBER_DATA	BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA	boost::tti::mtfc_has_static_member_data_'name'
BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA	BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA	boost::tti::'trait'
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION	BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION	boost::tti::mtfc_has_static_member_function_'name'
BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION	BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION	boost::tti::'trait'

Macro Metafunction	Metafunction Class Macro	Metafunction Class Name
BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION	BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION	boost::tti::mtfc_has_comp_static_member_function_'name'
BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION	BOOST_TTI_MTFC_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION	boost::tti::'trait'
BOOST_TTI_MEMBER_TYPE	BOOST_TTI_MTFC_MEMBER_TYPE	boost::tti::mtfc_member_type_'name'
BOOST_TTI_TRAIT_MEMBER_TYPE	BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE	boost::tti::'trait'

Each of the metafunction classes have exactly the same functionality as their corresponding macro metafunction. The metafunction class generated can be passed as metadata in the same way that their corresponding macro metafunctions with placeholder expressions can be passed as metadata.

As will be seen when we discuss the nullary type metafunctions, each nullary type metafunction takes as its first parameter a macro metafunction as metadata in the form of a lambda expression. Therefore we can use either a metafunction class generated by one of the macros above, or a placeholder expression, to pass a macro metafunction as metadata to our nullary type metafunctions. Other uses for our macro metafunctions as metadata may be found by the template metaprogrammer using the TTI library.

Nullary Type Metafunctions

The macro metafunctions provide a complete set of functionality for anything one would like to do using the TTI library. Why then do we have another set of parallel functionality as nullary type metafunctions ?

The nullary type metafunctions parallel only those macro metafunctions in which types are specified individually rather than in a composite manner. The individual types are passed to the nullary type metafunctions as nullary metafunctions, hence the name of this group of metafunctions. They more easily allow a syntax where nested types can be specified without needing to manually reach into the 'type' member of `BOOST_TTI_MEMBER_TYPE` or the 'type' member of its nullary type metafunction equivalent called `boost::tti::mf_member_type`.

In a very real way the nullary type metafunctions exist just to provide syntactic improvements over the macro metafunctions and are not needed to use the library, since all of the library functionality is already provided with the macro metafunctions. Nonetheless syntactic ease of use is a goal of the TTI library and therefore these metafunctions are provided to allow that syntactic improvement.

The nullary type metafunctions reuse the metafunctions generated by the macro metafunctions. To do this the resulting metafunction generated by a particular macro metafunction needs to be passed as metadata to a corresponding nullary type metafunction.

A lambda expression, in the form of a metafunction class or a placeholder expression, is passed as the first parameter to our nullary type metafunctions. We have already seen how the TTI library supplies metafunction classes through the use of metafunction class macros for each of the macro metafunctions. The end-user can use these metafunction classes directly, or can use placeholder expressions with the metafunctions generated by the macro metafunctions.

The remaining parameters passed to the nullary type metafunctions are 'types'. These 'types' always consist first of the enclosing type and then possibly other types which make up the signature of whatever inner element we are introspecting. Each of these 'types' is passed as a nullary metafunction whose typedef 'type' is the actual type.

The exception to this use of nullary type metafunctions when specifying 'types' is when a Boost `function_types` tag type, which is optional, is specified as an addition to the function signature. Also when dealing with a function signature and parameter types being passed, while the parameter 'types' themselves are in the form of nullary metafunctions, the MPL forward sequence which contains the parameter 'types' is a plain type and should not be wrapped as a nullary metafunction.

For a type which is in scope, we can always use boost::mpl::identity to create our nullary metafunction, and there can never be a compiler error for such known types as long as declarations for them exist or they are built-in C++ types. For nested types, which

may or may not exist, we can pass the resulting nullary metafunction generated by `BOOST_TTI_MEMBER_TYPE`, or its equivalent nullary type metafunction `boost::tti::mf_member_type` (explained later).

To use these metafunctions you need to include the main header file `tti.hpp`, unless otherwise noted. Alternatively you can include a specific header as given in the table below,

Tip

The header files <boost/mpl/identity.hpp> and <boost/mpl/placeholders.hpp> are included by the TTI header files whenever you include a general header file or a specific header file for a nullary type metafunction, so you need not manually include it in order to wrap a known type as a nullary metafunction or use a placeholder expression. Also the header file <boost/mpl/vector.hpp> is included by the general header file 'tti.hpp' or the specific header files which introspect functions, so if you use an MPL vector as your forward sequence wrapper for parameter types, you need not manually include the header file.

A table of these metafunctions is given, based on the inner element whose existence the metaprogrammer is introspecting. A more detailed explanation can be found in the reference section, and examples of usage can be found in the "Using the Nullary Type Metafunctions" section. All of the metafunctions are in the top-level 'boost::tti' namespace, all have a particular name based on the type of its functionality, and all begin with the prefix 'mf_' so as not to conflict with the macro metafunction names generated by the library.

Table 6. TTI Nullary Type Metafunctions

Inner Element	Template	Parameters	Macro Equivalent	Specific Header File
Type	boost::tti::mf_has_type	class HasType = macro metafunction as lambda expression class T = enclosing type nullary metafunction	BOOST_TTI_HAS_TYPE	type.hpp
Type with check	boost::tti::mf_has_type	class HasType = macro metafunction as lambda expression class T = enclosing type nullary metafunction class U = type to check against nullary metafunction	BOOST_TTI_HAS_TYPE	type.hpp
Class Template	boost::tti::mf_has_template	class HasTemplate = macro metafunction as lambda expression class T = enclosing type nullary metafunction	BOOST_TTI_HAS_TEMPLATE	template.hpp
Class Template with params	boost::tti::mf_has_template_check_params	class HasTemplateCheckParams = macro metafunction as lambda expression class T = enclosing type nullary metafunction	BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS	template_params.hpp vm_template_params.hpp
Member data	boost::tti::mf_has_member_data	class HasMemberData = macro metafunction as lambda expression class T = enclosing type nullary metafunction class R = type of member data nullary Metafunction	BOOST_TTI_HAS_MEMBER_DATA	mem_data.hpp

Inner Element	Template	Parameters	Macro Equivalent	Specific Header File
Member function	boost::tti::mf_has_member_function	class HasMemberFunction = macro metafunction as lambda expression class T = enclosing type nullary metafunction class R = return value nullary metafunction class FS = (optional) a Boost MPL forward sequence of parameter types as nullary metafunctions. The forward sequence as a type is not presented as a nullary metafunction. If there are no parameters, this may be omitted. class TAG = (optional) a Boost function_types tag type.	BOOST_TTI_HAS_MEMBER_FUNCTION	mem_fun.hpp
Static data	boost::tti::mf_has_static_member_data	class HasStaticMemberData = macro metafunction as lambda expression class T = enclosing type nullary metafunction class R = type of static data nullary metafunction	BOOST_TTI_HAS_STATIC_MEMBER_DATA	static_mem_data.hpp

Inner Element	Template	Parameters	Macro Equivalent	Specific Header File
Static function	boost::tti::mf_has_static_member_function	class HasStaticMemberFunction = macro metafunction as lambda expression class T = enclosing type nullary metafunction class R = return value nullary metafunction class FS = (optional) a Boost MPL forward sequence of parameter types as nullary metafunctions. The forward sequence as a type is not presented as a nullary metafunction. If there are no parameters, this may be omitted. class TAG = (optional) a Boost function_types tag type.	BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION	static_mem_fun.hpp

Other than the use of nullary metafunctions, one other difference in the nullary type metafunctions from their macro metafunction counterparts is that the signature for member functions and static member functions always involves individual types rather than the combined type notation which some of the macro metafunctions use. This allows us to specify nested types in those signatures without using the T::InnerType notation.

Nullary type metafunction member_type equivalent

Just as there exists the macro BOOST_TTI_MEMBER_TYPE for creating a macro metafunction which returns a nested type if it exists, else an unspecified type, there is also the equivalent nullary type metafunction.

Table 7. TTI Nested Type Nullary Type Metafunction

Inner Element	Template	Parameters	Macro Equivalent	Specific Header File
Type	boost::tti::mf_member_type	class MemberType = macro metafunction as lambda expression class T = enclosing type nullary metafunction	BOOST_TTI_MEMBER_TYPE	mem_type.hpp

The difference between the macro metafunction BOOST_TTI_MEMBER_TYPE and boost::tti::mf_member_type is simply that, like the other nullary type metafunctions, the latter takes its enclosing type as a nullary metafunction. Both produce the exact same result.

The use of this metafunction allows us to created deeply nested types, which may or may not exist, as nullary metafunctions in much the same way that BOOST_TTI_MEMBER_TYPE can. The difference is the simpler syntax when using boost::tti::mf_member_type.

As an example, given the theoretical relationship of types we used before:

```
struct T
  {
  struct AType
    {
    struct BType
      {
      struct CType
        {
        struct FindType
          {
          };
        }
      };
    };
  };
```

We can use `boost::tti::mf_member_type` as follows. First we create our corresponding macro metafunctions:

```
BOOST_TTI_MEMBER_TYPE(FindType)
BOOST_TTI_MEMBER_TYPE(AType)
BOOST_TTI_MEMBER_TYPE(BType)
BOOST_TTI_MEMBER_TYPE(CType)
```

Next we will create a typedef to reflect a nested type called FindType, as a nullary metafunction, which has the relationship as specified above, by using `boost::tti::mf_member_type`.

```
typedef
boost::tti::mf_member_type
  <
  boost::tti::member_type_FindType<_>,
  boost::tti::mf_member_type
    <
    boost::tti::member_type_CType<_>,
    boost::tti::mf_member_type
      <
      boost::tti::member_type_BType<_>,
      boost::tti::member_type_AType
        <
        T
        >
      >
    >
  > MyFindType;
```

The nested type created can then be used with the other nullary type metafunctions above since it is itself a nullary metafunction. The key information in understanding the code above is that the enclosing type, as in all of the nullary type metafunctions, is a nullary metafunction itself, which means that the enclosing type can be specified as the result of using `BOOST_TTI_MEMBER_TYPE` as well as the result of using `mf_member_type` itself.

Both techniques are shown in the example above, and the same technique for creating nested types as nullary metafunctions can be used with the other functionality of the nullary type metafunctions when nested types are needed as 'types'.

Also similar to the macro metafunctions, we have an easy way of testing whether or not our `boost::tti::mf_member_type` nested type actually exists.

Table 8. TTI Nested Type Nullary Type Metafunction Existence

Inner Element	Template	Parameters	Specific Header File
Type	`boost::tti::mf_valid_member_type`	class T = a type as a nullary metafunction returns = true if the nullary metafunction's inner 'type' exists, false if it does not. 'Existence' is determined by whether the type does not equal an unspecified type.	`mem_type.hpp`

Again note the difference here from the equivalent macro metafunction tester `boost::tti::valid_member_type`. In the example above the type T is passed as a nullary metafunction holding the actual type, where for the macro metafunction equivalent the type T is passed as the actual type being tested.

In our next section we will look at examples of nullary type metafunction use.

Using the Nullary Type Metafunctions

Using the nullary type metafunctions can be illustrated by first creating some hypothetical user-defined type with corresponding nested types and other inner elements. This user-defined type will be weighted toward showing deeply nested types and elements within those nested types. With this type we can illustrate the use of the nullary type metafunctions. This is just meant to serve as a model for what a type T might entail from within a class or function template.

```
// An enclosing type

struct T
  {

  // Type

  struct BType // as a nested type
    {

    // Template

    template <class,class,int,class,template <class> class InnerTem↵
plate,class,long> struct ManyParameters { };

    // Type

    struct CType // as a further nested type
      {

      // Template

      template <class> struct AMemberTemplate { };

      // Type

      struct DType // as a futher nested type
        {

        // Type

        typedef double ADoubleType;

        // Template

        template <class,class,int,short,class,template <class,int> class InnerTem↵
plate,class> struct MoreParameters { };

        // Function

        int IntFunction(short) const { return 0; }

        // Static Data

        static short DSMember;

        // Static Function

        static int SIntFunction(long,double) { return 2; }

        };
      };
    };

  // Data

  BType IntBT;

  };
```

I will be using the type above just to illustrate the sort of metaprogramming questions we can ask of some type T which is passed to the template programmer in a class template. Here is what the class template might look like:

```
#include <boost/tti/tti.hpp>

template<class T>
struct OurTemplateClass
  {

  // compile-time template code regarding T

  };
```

Now let us create and invoke the nested metafunctions for each of our inner element types, to see if type T above corresponds to our hypothetical type above. Imagine this being within 'OurTemplateClass' above. In the examples below the same macro is invoked just once to avoid ODR violations.

I will also be mixing the way the macro metafunction metadata is passed to our nullary type metafunctions, whether as a metafunction class or as a placeholder expression. Both will work just fine since our nullary type metafunctions work with any lambda expression as the first template parameter.

Member type

We start off by creating typedef's, as nullary metafunctions, for our theoretical inner types in relation to T . None of these typedefs will produce a compiler error even if our structure does not correspond to T's reality. This also illustrates using 'boost::tti::mf_member_type'.

```
BOOST_TTI_MEMBER_TYPE(BType)
BOOST_TTI_MTFC_MEMBER_TYPE(CType)
BOOST_TTI_MEMBER_TYPE(DType)

typedef
boost::tti::mf_member_type
  <
  boost::tti::member_type_BType<_>,
  boost::mpl::identity<T>
  >
BTypeNM;

typedef
boost::tti::mf_member_type
  <
  boost::tti::mtfc_member_type_CType,
  BTypeNM
  >
CTypeNM;

typedef
boost::tti::mf_member_type
  <
  boost::tti::member_type_DType<_>,
  CTypeNM
  >
DTypeNM;
```

We will use these typedefs in the ensuing examples.

Type

Does T have a nested type called 'DType' within 'BType::CType' ?

```
BOOST_TTI_HAS_TYPE(DType)

boost::tti::mf_has_type
  <
  boost::tti::has_type_DType<_>,
  CTypeNM
  >
```

We could just have easily used the `boost::tti::mf_valid_member_type` metafunction to the same effect:

```
boost::tti::mf_valid_member_type
  <
  DTypeNM
  >
```

Type checking the typedef

Does T have a nested typedef called 'ADoubleType' within 'BType::CType::DType' whose type is a 'double' ?

```
BOOST_TTI_MTFC_HAS_TYPE(ADoubleType)

boost::tti::mf_has_type
  <
  boost::tti::mtfc_has_type_ADoubleType,
  DTypeNM,
  boost::mpl::identity<double>
  >
```

Template

Does T have a nested class template called 'AMemberTemplate' within 'BType::CType' whose template parameters are all types ('class' or 'typename') ?

```
BOOST_TTI_MTFC_HAS_TEMPLATE(AMemberTemplate)

boost::tti::mf_has_template
  <
  boost::tti::mtfc_has_template_AMemberTemplate,
  CTypeNM
  >
```

Template with params

Does T have a nested class template called 'ManyParameters' within 'BType' whose template parameters are specified exactly ?

```
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS(ManyParameters,(class)(class)(int)(class)(tem↵
plate <class> class InnerTemplate)(class)(long))

boost::tti::mf_has_template_check_params
  <
  boost::tti::has_template_check_params_ManyParameters<_>,
  BTypeNM
  >
```

Template with params using variadic macros

Does T have a nested class template called 'MoreParameters' within 'BType::CType' whose template parameters are specified exactly ?

[note Include the `tti_vm.hpp` general header file, or the `vm_template_params.hpp` specific header file, when using this macro.]

```
BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS(MoreParameters,class,class,int,short,class,tem↵
plate <class,int> class InnerTemplate,class)

boost::tti::mf_has_template_check_params
  <
  boost::tti::mtfc_has_template_check_params_MoreParameters,
  CTypeNM
  >
```

Member data

Does T have a member data called 'IntBT' whose type is 'BType' ?

```
BOOST_TTI_MTFC_HAS_MEMBER_DATA(IntBT)

boost::tti::mf_has_member_data
  <
  boost::tti::mtfc_has_member_data_IntBT,
  boost::mpl::identity<T>,
  BTypeNM
  >
```

Member function

Does T have a member function called 'IntFunction' within 'BType::CType::DType' whose type is 'int (short) const' ?

```
BOOST_TTI_HAS_MEMBER_FUNCTION(IntFunction)

boost::tti::mf_has_member_function
  <
  boost::tti::has_member_function_IntFunction<_,_,_,_>,
  DTypeNM,
  boost::mpl::identity<int>,
  boost::mpl::vector<boost::mpl::identity<short> >,
  boost::function_types::const_qualified
  >
```

Static member data

Does T have a static member data called 'DSMember' within 'BType::CType::DType' whose type is 'short' ?

```
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA(DSMember)

boost::tti::mf_has_static_member_data
  <
  boost::tti::mtfc_has_static_member_data_DSMember,
  DTypeNM,
  boost::mpl::identity<short>
  >
```

Static member function

Does T have a static member function called 'SIntFunction' within 'BType::CType::DType' whose type is 'int (long,double)' ?

```
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION(SIntFunction)

boost::tti::mf_has_static_member_function
  <
  boost::tti::has_static_member_function_SIntFunction<_,_,_>,
  DTypeNM,
  boost::mpl::identity<int>,
  boost::mpl::vector<boost::mpl::identity<long>,boost::mpl::identity<double> >
  >
```

Introspecting Function Templates

The one nested element which the TTI library does not introspect is function templates.

Function templates, like functions, can be member function templates or static member function templates. In this respect they are related to functions. Function templates represent a family of possible functions. In this respect they are similar to class templates, which represent a family of possible class types.

The technique for introspecting class templates in the TTI library is taken from the implementation of the technique in the Boost MPL library. In the case of `BOOST_TTI_HAS_TEMPLATE` it directly uses the Boost MPL library functionality while in the case of `BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS` it replicates much of the technique in the Boost MPL library. The technique depends directly on the fact that in C++ we can pass a template as a parameter to another template using what is called a "template template" parameter type.

One obvious thing about a template template parameter type is that it is a class template. For whatever historical or technical reasons, no one has ever proposed that C++ have a way of passing a function template directly as a template parameter, perhaps to be called a "function template template" parameter type. I personally think this would be a good addition to C++ and would make the ability of passing a template as a parameter to another template more orthogonal, since both class templates and function templates would be supported. My efforts to discuss this on the major C++ newsgroups have met with arguments both against its practical usage and the justification that one can pass a function template to another template nested in a non-template class. But of course we can do the same thing with class templates, which is in fact what Boost MPL does to pass templates as metadata, yet we still have template template parameters as class templates.

Nonetheless the fact that we can pass class templates as a template parameter but not function templates as a template parameter is the major factor why there is no really good method for introspecting function templates at compile time.

Instantiating a nested function template

There is, however, an alternate but less certain way of introspecting a function template. I will endeavor to explain why this way is not currently included in the TTI library, but first I will explain what it is.

It is possible to check whether some particular instantiation of a nested function template exists at compile-time without generating a compiler error. While this does not prove that the nested function template does not exist, since the instantiation itself may be incorrect and fail even when the nested function template exists, it provides a partially flawed means of checking.

The code to do this for member function templates looks like this (similar code also exists for static member function templates):

```
template
  <
  class C,
  class T
  >
struct TestFunctionTemplate
  {
  typedef char Bad;
  struct Good { char x[2]; };
  template<T> struct helper;
  template<class U> static Good check(helper<&U::template SomeFuncTemplate↵
Name<int,long,double> > *);
  template<class U> static Bad check(...);
  static const bool value=sizeof(check<C>(0))==sizeof(Good);
  };
```

where 'SomeFuncTemplateName' is the name of the nested function template, followed by some parameters to instantiate it. The 'class T' is the type of the instantiated member function template as a member function, and 'class C' is the type of the enclosing class.

As an example if we had:

```
struct AType
  {
  template<class X,class Y,class Z> double SomeFuncTemplateName(X,Y *,Z &) { return 0.0; }
  };
```

then instantiating the above template with:

```
TestFunctionTemplate
  <
  AType,
  double (AType::*)(int,long *,double &)
  >
```

would provide a compile-time boolean value which would tell us whether the nested member function template exists for the particular instantiation provided above. Furthermore, through the use of a macro, the TTI library could provide the means for specifying the name of the nested member function template ('SomeFuncTemplateName' above) and its set of instantiated parameters ('int,long,double' above) for generating the template.

So why does not the TTI library not provide at least this much functionality for introspecting member function templates, even if it represents a partially flawed way of doing so ?

The reason is stunningly disappointing. Although the above code is perfectly correct C++ code ('clang' works correctly), two of the major C++ compilers, in all of their different releases, can not handle the above code correctly. Both gcc (g++) and Visual C++ incorrectly choose the wrong 'check' function even when the correct 'check' function applies (Comeau C++ also fails but I am less concerned about that compiler since it is not used nearly as much as the other two). All my attempts at alternatives to the above code have also failed. The problems with both compilers, in this regard, can be seen more easily with this snippet:

```
struct AType
 {
 template<class AA> void SomeFuncTemplate() { }
 };

template<class T>
struct Test
 {
 template<T> struct helper;
 template<class U> static void check(helper<&U::template SomeFuncTemplate<int> > *) { }
 };

int main()
  {
  Test< void (AType::*)() >::check<AType>(0);
  return 0;
  }
```

Both compilers report compile errors with this perfectly correct code,

gcc:

```
error: no matching function for call to 'Test<void (AType::*)()>::check(int)'
```

and msvc:

```
error C2770: invalid explicit template argument(s) for 'void Test<T>::check(Test<T>::help↵
er<&U::SomeFuncTemplate<int>> *)'
```

There is a workaround for these compiler problems, which is to hardcode the name of the enclosing class, via a macro, in the generated template rather than pass it as a template type. In that case both compilers can handle both the member function code and the code snippet above correctly. In essence, when the line:

```
template<class U> static void check(helper<&U::template SomeFuncTemplate<int> > *) { }
```

gets replaced by:

```
template<class U> static void check(helper<&AType::template SomeFuncTemplate<int> > *) { }
```

both gcc and Visual C++ work correctly. The same goes for the 'check' line in the 'TestFunctionTemplate' above.

But the workaround destroys one of the basic tenets of the TTI library, which is that the enclosing class be passed as a template parameter, especially as the enclosing class need not actually exist (see `BOOST_TTI_MEMBER_TYPE` and the previous discussion of 'Nested Types'), without producing a compiler error. So I have decided not to implement even this methodology to introspect nested function templates in the TTI library.

Type Traits Introspection Reference

Header <boost/tti/comp_mem_fun.hpp>

```
BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_COMP_MEMBER_FUNCTION(trait, name)
BOOST_TTI_HAS_COMP_MEMBER_FUNCTION(name)
BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION(name)
```

Macro BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION

BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION — Expands to a metafunction which tests whether a member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_mem_fun.hpp>

BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner member.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the member function type, in the form of a member function pointer - 'return_type (Class::*)(parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_COMP_MEMBER_FUNCTION

BOOST_TTI_MTFC_TRAIT_HAS_COMP_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_mem_fun.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_COMP_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner member.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the member function type, in the form of a member function pointer - 'return_type (Class::*)(parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_HAS_COMP_MEMBER_FUNCTION

BOOST_TTI_HAS_COMP_MEMBER_FUNCTION — Expands to a metafunction which tests whether a member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_mem_fun.hpp>

BOOST_TTI_HAS_COMP_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction called "boost::tti::has_member_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the member function type, in the form of a member function pointer - 'return_type (Class::*)(parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION

BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_mem_fun.hpp>

BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction class called "boost::tti::mtfc_has_member_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the member function type, in the form of a member function pointer - 'return_type (Class::*)(parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Header <boost/tti/comp_static_mem_fun.hpp>

```


BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION(trait, name)
BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION(name)
BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION(name)
```

Macro BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION

BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION — Expands to a metafunction which tests whether a static member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_static_mem_fun.hpp>

BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner member.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type.

Type = the static member function type, in the form of a composite function type - 'return_type (parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION

BOOST_TTI_MTFC_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a static member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_static_mem_fun.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner member.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type.

Type = the static member function type, in the form of a composite function type - 'return_type (parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION

BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION — Expands to a metafunction which tests whether a static member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_static_mem_fun.hpp>

BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction called "boost::tti::has_static_member_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type.

Type = the static member function type, in the form of a composite function type - 'return_type (parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION

BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a static member function with a particular name and composite type exists.

Synopsis

```
// In header: <boost/tti/comp_static_mem_fun.hpp>

BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction class called "boost::tti::mtfc_has_static_member_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type.

Type = the static member function type, in the form of a composite function type - 'return_type (parameter_types...)', in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Header <boost/tti/gen/comp_mem_fun_gen.hpp>

```
BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN(name)
BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN(name)
```

Macro BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_COMP_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_mem_fun_gen.hpp>

BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN

BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_COMP_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_mem_fun_gen.hpp>

BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN

BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Header <boost/tti/gen/comp_static_mem_fun_gen.hpp>

```


BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN(name)
BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN(name)
```

Macro BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_static_mem_fun_gen.hpp>

BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_static_mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN

BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_static_mem_fun_gen.hpp>

BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN

BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/comp_static_mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Header <boost/tti/gen/mem_data_gen.hpp>

```


BOOST_TTI_HAS_MEMBER_DATA_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN_BASE(name)
BOOST_TTI_HAS_MEMBER_DATA_GEN(name)
BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN(name)
```

Macro BOOST_TTI_HAS_MEMBER_DATA_GEN_BASE

BOOST_TTI_HAS_MEMBER_DATA_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/mem_data_gen.hpp>

BOOST_TTI_HAS_MEMBER_DATA_GEN_BASE(name)
```

Description

name = the name of the member data.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN_BASE

BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/mem_data_gen.hpp>

BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN_BASE(name)
```

Description

name = the name of the member data.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_MEMBER_DATA_GEN

BOOST_TTI_HAS_MEMBER_DATA_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/mem_data_gen.hpp>

BOOST_TTI_HAS_MEMBER_DATA_GEN(name)
```

Description

name = the name of the member data.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN

BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/mem_data_gen.hpp>

BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN(name)
```

Description

name = the name of the member data.

returns = the generated macro metafunction name.

Header <boost/tti/gen/mem_fun_gen.hpp>

```


BOOST_TTI_HAS_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_HAS_MEMBER_FUNCTION_GEN(name)
BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN(name)
```

Macro BOOST_TTI_HAS_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_HAS_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/mem_fun_gen.hpp>

BOOST_TTI_HAS_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_MEMBER_FUNCTION_GEN

BOOST_TTI_HAS_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/mem_fun_gen.hpp>

BOOST_TTI_HAS_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN

BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the member function.

returns = the generated macro metafunction name.

Header <boost/tti/gen/mem_type_gen.hpp>

```


BOOST_TTI_MEMBER_TYPE_GEN_BASE(name)
BOOST_TTI_MTFC_MEMBER_TYPE_GEN_BASE(name)
BOOST_TTI_MEMBER_TYPE_GEN(name)
BOOST_TTI_MTFC_MEMBER_TYPE_GEN(name)
```

Macro BOOST_TTI_MEMBER_TYPE_GEN_BASE

BOOST_TTI_MEMBER_TYPE_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MEMBER_TYPE.

Synopsis

```
// In header: <boost/tti/gen/mem_type_gen.hpp>

BOOST_TTI_MEMBER_TYPE_GEN_BASE(name)
```

Description

name = the name of the inner type.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_MEMBER_TYPE_GEN_BASE

BOOST_TTI_MTFC_MEMBER_TYPE_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_MEMBER_TYPE.

Synopsis

```
// In header: <boost/tti/gen/mem_type_gen.hpp>

BOOST_TTI_MTFC_MEMBER_TYPE_GEN_BASE(name)
```

Description

name = the name of the inner type.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MEMBER_TYPE_GEN

BOOST_TTI_MEMBER_TYPE_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MEMBER_TYPE.

Synopsis

```
// In header: <boost/tti/gen/mem_type_gen.hpp>

BOOST_TTI_MEMBER_TYPE_GEN(name)
```

Description

name = the name of the inner type.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_MEMBER_TYPE_GEN

BOOST_TTI_MTFC_MEMBER_TYPE_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_MEMBER_TYPE.

Synopsis

```
// In header: <boost/tti/gen/mem_type_gen.hpp>

BOOST_TTI_MTFC_MEMBER_TYPE_GEN(name)
```

Description

name = the name of the inner type.

returns = the generated macro metafunction name.

Header <boost/tti/gen/namespace_gen.hpp>

```


BOOST_TTI_NAMESPACE
```

Macro BOOST_TTI_NAMESPACE

BOOST_TTI_NAMESPACE — Generates the name of the Boost TTI namespace.

Synopsis

```
// In header: <boost/tti/gen/namespace_gen.hpp>

BOOST_TTI_NAMESPACE
```

Description

returns = the generated name of the Boost TTI namespace.

Header <boost/tti/gen/static_mem_data_gen.hpp>

```


BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN_BASE(name)
BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN(name)
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN(name)
```

Macro BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN_BASE

BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_STATIC_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/static_mem_data_gen.hpp>

BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN_BASE(name)
```

Description

name = the name of the static member data.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN_BASE

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/static_mem_data_gen.hpp>

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN_BASE(name)
```

Description

name = the name of the static member data.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN

BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_STATIC_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/static_mem_data_gen.hpp>

BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN(name)
```

Description

name = the name of the static member data.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA.

Synopsis

```
// In header: <boost/tti/gen/static_mem_data_gen.hpp>

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN(name)
```

Description

name = the name of the static member data.

returns = the generated macro metafunction name.

Header <boost/tti/gen/static_mem_fun_gen.hpp>

```


BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN(name)
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN(name)
```

Macro BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/static_mem_fun_gen.hpp>

BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/static_mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN

BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/static_mem_fun_gen.hpp>

BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION.

Synopsis

```
// In header: <boost/tti/gen/static_mem_fun_gen.hpp>

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN(name)
```

Description

name = the name of the static member function.

returns = the generated macro metafunction name.

Header <boost/tti/gen/template_gen.hpp>

```


BOOST_TTI_HAS_TEMPLATE_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_TEMPLATE_GEN_BASE(name)
BOOST_TTI_HAS_TEMPLATE_GEN(name)
BOOST_TTI_MTFC_HAS_TEMPLATE_GEN(name)
```

Macro BOOST_TTI_HAS_TEMPLATE_GEN_BASE

BOOST_TTI_HAS_TEMPLATE_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_TEMPLATE.

Synopsis

```
// In header: <boost/tti/gen/template_gen.hpp>

BOOST_TTI_HAS_TEMPLATE_GEN_BASE(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_TEMPLATE_GEN_BASE

BOOST_TTI_MTFC_HAS_TEMPLATE_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_TEMPLATE.

Synopsis

```
// In header: <boost/tti/gen/template_gen.hpp>

BOOST_TTI_MTFC_HAS_TEMPLATE_GEN_BASE(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_TEMPLATE_GEN

BOOST_TTI_HAS_TEMPLATE_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_TEMPLATE.

Synopsis

```
// In header: <boost/tti/gen/template_gen.hpp>

BOOST_TTI_HAS_TEMPLATE_GEN(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_TEMPLATE_GEN

BOOST_TTI_MTFC_HAS_TEMPLATE_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_TEMPLATE.

Synopsis

```
// In header: <boost/tti/gen/template_gen.hpp>

BOOST_TTI_MTFC_HAS_TEMPLATE_GEN(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Header <boost/tti/gen/template_params_gen.hpp>

```


BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
```

Macro BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE

BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/template_params_gen.hpp>

BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE

BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/template_params_gen.hpp>

BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN

BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/template_params_gen.hpp>

BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN

BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/template_params_gen.hpp>

BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Header <boost/tti/gen/trait_gen.hpp>

```


BOOST_TTI_TRAIT_GEN_BASE(trait)
BOOST_TTI_TRAIT_GEN(trait)
```

Macro BOOST_TTI_TRAIT_GEN_BASE

BOOST_TTI_TRAIT_GEN_BASE — Generates the macro metafunction name for any macro metafunction using the 'trait' form.

Synopsis

```
// In header: <boost/tti/gen/trait_gen.hpp>

BOOST_TTI_TRAIT_GEN_BASE(trait)
```

Description

trait = the name of the trait.

returns = the generated macro metafunction name.

Macro BOOST_TTI_TRAIT_GEN

BOOST_TTI_TRAIT_GEN — Generates the macro metafunction name within the Boost TTI namespace for any macro metafunction using the 'trait' form.

Synopsis

```
// In header: <boost/tti/gen/trait_gen.hpp>

BOOST_TTI_TRAIT_GEN(trait)
```

Description

trait = the name of the trait.

returns = the generated macro metafunction name.

Header <boost/tti/gen/type_gen.hpp>

```


BOOST_TTI_HAS_TYPE_GEN_BASE(name)
BOOST_TTI_MTFC_HAS_TYPE_GEN_BASE(name)
BOOST_TTI_HAS_TYPE_GEN(name)
BOOST_TTI_MTFC_HAS_TYPE_GEN(name)
```

Macro BOOST_TTI_HAS_TYPE_GEN_BASE

BOOST_TTI_HAS_TYPE_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_HAS_TYPE.

Synopsis

```
// In header: <boost/tti/gen/type_gen.hpp>

BOOST_TTI_HAS_TYPE_GEN_BASE(name)
```

Description

name = the name of the type.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_TYPE_GEN_BASE

BOOST_TTI_MTFC_HAS_TYPE_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_MTFC_HAS_TYPE.

Synopsis

```
// In header: <boost/tti/gen/type_gen.hpp>

BOOST_TTI_MTFC_HAS_TYPE_GEN_BASE(name)
```

Description

name = the name of the type.

returns = the generated macro metafunction name.

Macro BOOST_TTI_HAS_TYPE_GEN

BOOST_TTI_HAS_TYPE_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_HAS_TYPE.

Synopsis

```
// In header: <boost/tti/gen/type_gen.hpp>

BOOST_TTI_HAS_TYPE_GEN(name)
```

Description

name = the name of the type.

returns = the generated macro metafunction name.

Macro BOOST_TTI_MTFC_HAS_TYPE_GEN

BOOST_TTI_MTFC_HAS_TYPE_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_MTFC_HAS_TYPE.

Synopsis

```
// In header: <boost/tti/gen/type_gen.hpp>

BOOST_TTI_MTFC_HAS_TYPE_GEN(name)
```

Description

name = the name of the type.

returns = the generated macro metafunction name.

Header <boost/tti/gen/vm_template_params_gen.hpp>

```


BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
```

Macro BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE

BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/vm_template_params_gen.hpp>

BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE

BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE — Generates the macro metafunction name for BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/vm_template_params_gen.hpp>

BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN

BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/vm_template_params_gen.hpp>

BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Macro BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN

BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN — Generates the macro metafunction name within the Boost TTI namespace for BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS.

Synopsis

```
// In header: <boost/tti/gen/vm_template_params_gen.hpp>

BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN(name)
```

Description

name = the name of the class template.

returns = the generated macro metafunction name.

Header <boost/tti/mem_data.hpp>

```


BOOST_TTI_TRAIT_HAS_MEMBER_DATA(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_DATA(trait, name)
BOOST_TTI_HAS_MEMBER_DATA(name)
BOOST_TTI_MTFC_HAS_MEMBER_DATA(name)
```

Macro BOOST_TTI_TRAIT_HAS_MEMBER_DATA

BOOST_TTI_TRAIT_HAS_MEMBER_DATA — Expands to a metafunction which tests whether a member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/mem_data.hpp>

BOOST_TTI_TRAIT_HAS_MEMBER_DATA(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner member.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the type of the member data.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_DATA

BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_DATA — Expands to a metafunction class which tests whether a member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/mem_data.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_DATA(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner member.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the type of the member data.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_HAS_MEMBER_DATA

BOOST_TTI_HAS_MEMBER_DATA — Expands to a metafunction which tests whether a member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/mem_data.hpp>

BOOST_TTI_HAS_MEMBER_DATA(name)
```

Description

name = the name of the inner member.

returns = a metafunction called "boost::tti::has_member_data_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the type of the member data.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_MEMBER_DATA

BOOST_TTI_MTFC_HAS_MEMBER_DATA — Expands to a metafunction class which tests whether a member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/mem_data.hpp>

BOOST_TTI_MTFC_HAS_MEMBER_DATA(name)
```

Description

name = the name of the inner member.

returns = a metafunction class called "boost::tti::mtfc_has_member_data_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the type of the member data.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Header <boost/tti/mem_fun.hpp>

```


BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION(trait, name)
BOOST_TTI_HAS_MEMBER_FUNCTION(name)
BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION(name)
```

Macro BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION

BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION — Expands to a metafunction which tests whether a member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/mem_fun.hpp>

BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner member.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the member function.

FS = an optional parameter which are the parameters of the member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION

BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/mem_fun.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner member.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the member function.

FS = an optional parameter which are the parameters of the member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_HAS_MEMBER_FUNCTION

BOOST_TTI_HAS_MEMBER_FUNCTION — Expands to a metafunction which tests whether a member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/mem_fun.hpp>

BOOST_TTI_HAS_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction called "boost::tti::has_member_function_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the member function.

FS = an optional parameter which are the parameters of the member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION

BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/mem_fun.hpp>

BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction class called "boost::tti::mtfc_has_member_function_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the member function.

FS = an optional parameter which are the parameters of the member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Header <boost/tti/mem_type.hpp>

```


BOOST_TTI_TRAIT_MEMBER_TYPE(trait, name)
BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE(trait, name)
BOOST_TTI_MEMBER_TYPE(name)
BOOST_TTI_MTFC_MEMBER_TYPE(name)
```

Macro BOOST_TTI_TRAIT_MEMBER_TYPE

BOOST_TTI_TRAIT_MEMBER_TYPE — Expands to a metafunction whose typedef 'type' is either the named type or an unspecified type.

Synopsis

```
// In header: <boost/tti/mem_type.hpp>

BOOST_TTI_TRAIT_MEMBER_TYPE(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner type.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type.

returns = 'type' is the inner type of 'name' if the inner type exists within the enclosing type, else 'type' is an unspecified type.

The purpose of this macro is to encapsulate the 'name' type as the typedef 'type' of a metafunction, but only if it exists within the enclosing type. This allows for a lazy evaluation of inner type existence which can be used by other metafunctions in this library.

Macro BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE

BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE — Expands to a metafunction class whose typedef 'type' is either the named type or an unspecified type.

Synopsis

```
// In header: <boost/tti/mem_type.hpp>

BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner type.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type.

returns = 'type' is the inner type of 'name' if the inner type exists within the enclosing type, else 'type' is an unspecified type.

The purpose of this macro is to encapsulate the 'name' type as the typedef 'type' of a metafunction class, but only if it exists within the enclosing type. This allows for a lazy evaluation of inner type existence which can be used by other metafunctions in this library.

Macro BOOST_TTI_MEMBER_TYPE

BOOST_TTI_MEMBER_TYPE — Expands to a metafunction whose typedef 'type' is either the named type or an unspecified type.

Synopsis

```
// In header: <boost/tti/mem_type.hpp>

BOOST_TTI_MEMBER_TYPE(name)
```

Description

name = the name of the inner type.

returns = a metafunction called "boost::tti::member_type_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type.

returns = 'type' is the inner type of 'name' if the inner type exists within the enclosing type, else 'type' is an unspecified type.

The purpose of this macro is to encapsulate the 'name' type as the typedef 'type' of a metafunction, but only if it exists within the enclosing type. This allows for a lazy evaluation of inner type existence which can be used by other metafunctions in this library.

Macro BOOST_TTI_MTFC_MEMBER_TYPE

BOOST_TTI_MTFC_MEMBER_TYPE — Expands to a metafunction class whose typedef 'type' is either the named type or an unspecified type.

Synopsis

```
// In header: <boost/tti/mem_type.hpp>

BOOST_TTI_MTFC_MEMBER_TYPE(name)
```

Description

name = the name of the inner type.

returns = a metafunction class called "boost::tti::mtfc_member_type_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type.

returns = 'type' is the inner type of 'name' if the inner type exists within the enclosing type, else 'type' is an unspecified type.

The purpose of this macro is to encapsulate the 'name' type as the typedef 'type' of a metafunction class, but only if it exists within the enclosing type. This allows for a lazy evaluation of inner type existence which can be used by other metafunctions in this library.

Header <boost/tti/mf/mf_mem_data.hpp>

```
namespace boost {
  namespace tti {
    template<typename HasMemberData, typename T, typename R>
      struct mf_has_member_data;
  }
}
```

Struct template mf_has_member_data

boost::tti::mf_has_member_data — A metafunction which checks whether a member data exists within an enclosing type.

Synopsis

```
// In header: <boost/tti/mf/mf_mem_data.hpp>

template<typename HasMemberData, typename T, typename R>
struct mf_has_member_data {
};
```

Description

This metafunction takes its specific types as nullary metafunctions whose typedef 'type' member is the actual type used.

The metafunction types and return:

HasMemberData = a Boost MPL lambda expression using the metafunction generated from the TTI_HAS_MEMBER_DATA (or TTI_TRAIT_HAS_MEMBER_DATA) macro.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_,_>'. You can also use the metafunction class generated by the TTI_MTFC_HAS_MEMBER_DATA (or TTI_MTFC_TRAIT_HAS_MEMBER_DATA) macro.

T = the enclosing type as a nullary metafunction.

R = the type of the member data as a nullary metafunction.

returns = 'value' is true if the member data exists within the enclosing type, otherwise 'value' is false.

Header <boost/tti/mf/mf_mem_fun.hpp>

```
namespace boost {
  namespace tti {
    template<typename HasMemberFunction, typename T, typename R,
             typename FS = boost::mpl::vector<>,
             typename TAG = boost::function_types::null_tag>
      struct mf_has_member_function;
  }
}
```

Struct template mf_has_member_function

boost::tti::mf_has_member_function — A metafunction which checks whether a member function exists within an enclosing type.

Synopsis

```
// In header: <boost/tti/mf/mf_mem_fun.hpp>

template<typename HasMemberFunction, typename T, typename R,
         typename FS = boost::mpl::vector<>,
         typename TAG = boost::function_types::null_tag>
struct mf_has_member_function {
};
```

Description

This metafunction takes its specific types, except for the optional parameters, as nullary metafunctions whose typedef 'type' member is the actual type used.

The metafunction types and return:

HasMemberFunction = a Boost MPL lambda expression using the metafunction generated from the TTI_HAS_MEMBER_FUNCTION (or TTI_TRAIT_HAS_MEMBER_FUNCTION) macro.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_,_> (or optionally 'metafunction<_,_,_> or ' 'metafunction<_,_,_,_>)'. You can also use the metafunction class generated by the TTI_MTFC_HAS_MEMBER_FUNCTION (or TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION) macro.

T = the enclosing type as a nullary metafunction.

R = the return type of the member function as a nullary metafunction.

FS = an optional parameter which is the parameters of the member function, each as a nullary metafunction, as a boost::mpl forward sequence.
This parameter defaults to boost::mpl::vector<>.

TAG = an optional parameter which is a boost::function_types tag to apply to the member function.
This parameter defaults to boost::function_types::null_tag.

returns = 'value' is true if the member function exists within the enclosing type, otherwise 'value' is false.

Header <boost/tti/mf/mf_mem_type.hpp>

```
namespace boost {
  namespace tti {
    template<typename T> struct valid_member_type;
    template<typename T> struct mf_valid_member_type;
    template<typename MemberType, typename T> struct mf_member_type;
  }
}
```

Struct template valid_member_type

boost::tti::valid_member_type — A metafunction which checks whether the member 'type' returned from invoking the macro metafunction generated by TTI_MEMBER_TYPE (TTI_TRAIT_MEMBER_TYPE) or from invoking boost::tti::mf_member_type is a valid type.

Synopsis

```
// In header: <boost/tti/mf/mf_mem_type.hpp>

template<typename T>
struct valid_member_type {
};
```

Description

The metafunction types and return:

T = returned inner 'type' from invoking the macro metafunction generated by TTI_MEMBER_TYPE (TTI_TRAIT_MEMBER_TYPE) or from invoking boost::tti::mf_member_type.

returns = 'value' is true if the type is valid, otherwise 'value' is false.

Struct template mf_valid_member_type

boost::tti::mf_valid_member_type — A metafunction which checks whether the member 'type' returned from invoking the macro metafunction generated by TTI_MEMBER_TYPE (TTI_TRAIT_MEMBER_TYPE) or from invoking boost::tti::mf_member_type is a valid type.

Synopsis

```
// In header: <boost/tti/mf/mf_mem_type.hpp>

template<typename T>
struct mf_valid_member_type {
};
```

Description

The metafunction types and return:

T = the nullary metafunction from invoking the macro metafunction generated by TTI_MEMBER_TYPE (TTI_TRAIT_MEMBER_TYPE) or from invoking boost::tti::mf_member_type.

returns = 'value' is true if the type is valid, otherwise 'value' is false.

Struct template mf_member_type

boost::tti::mf_member_type — A metafunction whose typedef 'type' is either the internal type or an unspecified type.

Synopsis

```
// In header: <boost/tti/mf/mf_mem_type.hpp>

template<typename MemberType, typename T>
struct mf_member_type {
};
```

Description

This metafunction takes its enclosing type as a nullary metafunction whose typedef 'type' member is the actual type used.

The metafunction types and return:

MemberType = a Boost MPL lambda expression using the metafunction generated from the TTI_MEMBER_TYPE (or TTI_TRAIT_MEMBER_TYPE) macro.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_>'. You can also use the metafunction class generated by the TTI_MTFC_MEMBER_TYPE (or TTI_MTFC_TRAIT_MEMBER_TYPE) macro.

T = the enclosing type as a nullary metafunction.

returns = 'type' is the inner type of the 'name' in TTI_MEMBER_TYPE (or TTI_TRAIT_MEMBER_TYPE) if the inner type exists within the enclosing type, else 'type' is an unspecified type.

The purpose of this metafunction is to encapsulate the 'name' type in TTI_MEMBER_TYPE (or TTI_TRAIT_MEMBER_TYPE) as the typedef 'type' of a metafunction, but only if it exists within the enclosing type. This allows for a lazy evaluation of inner type existence which can be used by other metafunctions in this library.

Furthermore this metafunction allows the enclosing type to be return type from either the metafunction generated from TTI_MEMBER_TYPE (or TTI_TRAIT_MEMBER_TYPE) or from this metafunction itself.

Header <boost/tti/mf/mf_static_mem_data.hpp>

```
namespace boost {
  namespace tti {
    template<typename HasStaticMemberData, typename T, typename R>
      struct mf_has_static_member_data;
  }
}
```

Struct template mf_has_static_member_data

boost::tti::mf_has_static_member_data — A metafunction which checks whether a static member data exists within an enclosing type.

Synopsis

```
// In header: <boost/tti/mf/mf_static_mem_data.hpp>

template<typename HasStaticMemberData, typename T, typename R>
struct mf_has_static_member_data {
};
```

Description

This metafunction takes its specific types as nullary metafunctions whose typedef 'type' member is the actual type used.

The metafunction types and return:

HasStaticMemberData = a Boost MPL lambda expression using the metafunction generated from the TTI_HAS_STATIC_MEMBER_DATA (or TTI_TRAIT_HAS_STATIC_MEMBER_DATA) macro.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_,_>'. You can also use the metafunction class generated by the TTI_MTFC_HAS_STATIC_MEMBER_DATA (or TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA) macro.

T = the enclosing type as a nullary metafunction.

R = the type of the static member data as a nullary metafunction.

returns = 'value' is true if the member data exists within the enclosing type, otherwise 'value' is false.

Header <boost/tti/mf/mf_static_mem_fun.hpp>

```
namespace boost {
  namespace tti {
    template<typename HasStaticMemberFunction, typename T, typename R,
             typename FS = boost::mpl::vector<>,
             typename TAG = boost::function_types::null_tag>
      struct mf_has_static_member_function;
  }
}
```

Struct template mf_has_static_member_function

boost::tti::mf_has_static_member_function — A metafunction which checks whether a static member function exists within an enclosing type.

Synopsis

```
// In header: <boost/tti/mf/mf_static_mem_fun.hpp>

template<typename HasStaticMemberFunction, typename T, typename R,
         typename FS = boost::mpl::vector<>,
         typename TAG = boost::function_types::null_tag>
struct mf_has_static_member_function {
};
```

Description

This metafunction takes its specific types, except for the optional parameters, as nullary metafunctions whose typedef 'type' member is the actual type used.

The metafunction types and return:

HasStaticMemberFunction = a Boost MPL lambda expression using the metafunction generated from the TTI_HAS_STATIC_MEMBER_FUNCTION (or TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION) macro.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_,_> (or optionally 'metafunction<_,_,_> or ' 'metafunction<_,_,_,_>)'. You can also use the metafunction class generated by the TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION (or TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION) macro.

T = the enclosing type as a nullary metafunction.

R = the return type of the static member function as a nullary metafunction.

FS = an optional parameter which is the parameters of the static member function, each as a nullary metafunction, as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the static member function.

returns = 'value' is true if the member function exists within the enclosing type, otherwise 'value' is false.

Header <boost/tti/mf/mf_template.hpp>

```
namespace boost {
  namespace tti {
    template<typename HasTemplate, typename T> struct mf_has_template;
  }
}
```

Struct template mf_has_template

boost::tti::mf_has_template — A metafunction which checks whether a class template exists within an enclosing type.

Synopsis

```
// In header: <boost/tti/mf/mf_template.hpp>

template<typename HasTemplate, typename T>
struct mf_has_template {
};
```

Description

This metafunction takes its enclosing type as nullary metafunctions whose typedef 'type' member is the actual type used.

The metafunction types and return:

HasTemplate = a Boost MPL lambda expression using the metafunction generated from the TTI_HAS_TEMPLATE (TTI_TRAIT_HAS_TEMPLATE) macro.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_>'. You can also use the metafunction class generated by the TTI_MTFC_HAS_TEMPLATE (TTI_MTFC_TRAIT_HAS_TEMPLATE) macro.

T = the enclosing type as a nullary metafunction.

returns = 'value' is true if the template exists within the enclosing type, otherwise 'value' is false.

Header <boost/tti/mf/mf_template_params.hpp>

```
namespace boost {
  namespace tti {
    template<typename HasTemplateCheckParams, typename T>
      struct mf_has_template_check_params;
  }
}
```

Struct template mf_has_template_check_params

boost::tti::mf_has_template_check_params — A metafunction which checks whether a class template with its parameters exists within an enclosing type.

Synopsis

```
// In header: <boost/tti/mf/mf_template_params.hpp>

template<typename HasTemplateCheckParams, typename T>
struct mf_has_template_check_params {
};
```

Description

This metafunction takes its enclosing type as nullary metafunctions whose typedef 'type' member is the actual type used.

The metafunction types and return:

HasTemplateCheckParams = a Boost MPL lambda expression using the metafunction generated from either the TTI_HAS_TEMPLATE_CHECK_PARAMS (TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS) or TTI_VM_HAS_TEMPLATE_CHECK_PARAMS (TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS) macros.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_>'. You can also use the metafunction class generated by either the TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS (TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS) macro or the TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS (TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS) macro.

T = The enclosing type as a nullary metafunction.

returns = 'value' is true if the template exists within the enclosing type, otherwise 'value' is false.

Header <boost/tti/mf/mf_type.hpp>

```
namespace boost {
  namespace tti {
    template<typename HasType, typename T,
             typename U = boost::mpl::identity<BOOST_TTI_NAMESPACE::detail::notype> >
      struct mf_has_type;
  }
}
```

Struct template mf_has_type

boost::tti::mf_has_type — A metafunction which checks whether a type exists within an enclosing type and optionally is a particular type.

Synopsis

```
// In header: <boost/tti/mf/mf_type.hpp>

template<typename HasType, typename T,
         typename U = boost::mpl::identity<BOOST_TTI_NAMESPACE::detail::notype> >
struct mf_has_type {
};
```

Description

This metafunction takes its specific types as nullary metafunctions whose typedef 'type' member is the actual type used.

The metafunction types and return:

HasType = a Boost MPL lambda expression using the metafunction generated from the TTI_HAS_TYPE (or TTI_TRAIT_HAS_TYPE) macro.
The easiest way to generate the lambda expression is to use a Boost MPL placeholder expression of the form 'metafunction<_>' (or optionally 'metafunction<_,_>'). You can also use the metafunction class generated by the TTI_MTFC_HAS_TYPE (or TTI_MTFC_TRAIT_HAS_TYPE) macro.

T = the enclosing type as a nullary metafunction.

U = the type of the inner type as a nullary metafunction, as an optional parameter.

returns = 'value' is true if the type exists within the enclosing type and, if type U is specified, the type is the same as the type U, otherwise 'value' is false.

Header <boost/tti/static_mem_data.hpp>

```
BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA(trait, name)
BOOST_TTI_HAS_STATIC_MEMBER_DATA(name)
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA(name)
```

Macro BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA

BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA — Expands to a metafunction which tests whether a static member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/static_mem_data.hpp>

BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner member.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type.

Type = the static member data type, in the form of a data type, in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA

BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA — Expands to a metafunction class which tests whether a static member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/static_mem_data.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner member.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type.

Type = the static member data type, in the form of a data type, in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_HAS_STATIC_MEMBER_DATA

BOOST_TTI_HAS_STATIC_MEMBER_DATA — Expands to a metafunction which tests whether a static member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/static_mem_data.hpp>

BOOST_TTI_HAS_STATIC_MEMBER_DATA(name)
```

Description

name = the name of the inner member.

returns = a metafunction called "boost::tti::has_static_member_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type.

Type = the static member data type, in the form of a data type, in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA — Expands to a metafunction class which tests whether a static member data with a particular name and type exists.

Synopsis

```
// In header: <boost/tti/static_mem_data.hpp>

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA(name)
```

Description

name = the name of the inner member.

returns = a metafunction class called "boost::tti::mtfc_has_static_member_data_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type.

Type = the static member data type, in the form of a data type, in which to look for our 'name'.

returns = 'value' is true if the 'name' exists within the enclosing type, with the appropriate type, otherwise 'value' is false.

Header <boost/tti/static_mem_fun.hpp>

```
BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION(trait, name)
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION(name)
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION(name)
```

Macro BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION

BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION — Expands to a metafunction which tests whether a static member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/static_mem_fun.hpp>

BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner member.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the static member function.

FS = an optional parameter which are the parameters of the static member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the static member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION

BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a static member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/static_mem_fun.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner member.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the static member function.

FS = an optional parameter which are the parameters of the static member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the static member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION

BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION — Expands to a metafunction which tests whether a static member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/static_mem_fun.hpp>

BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction called "boost::tti::has_static_member_function_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the static member function.

FS = an optional parameter which are the parameters of the static member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the static member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION — Expands to a metafunction class which tests whether a static member function with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/static_mem_fun.hpp>

BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION(name)
```

Description

name = the name of the inner member.

returns = a metafunction class called "boost::tti::mtfc_has_static_member_function_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

R = the return type of the static member function.

FS = an optional parameter which are the parameters of the static member function as a boost::mpl forward sequence.

TAG = an optional parameter which is a boost::function_types tag to apply to the static member function.

returns = 'value' is true if the 'name' exists, with the appropriate type, otherwise 'value' is false.

Header <boost/tti/template.hpp>

```


BOOST_TTI_TRAIT_HAS_TEMPLATE(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE(trait, name)
BOOST_TTI_HAS_TEMPLATE(name)
BOOST_TTI_MTFC_HAS_TEMPLATE(name)
```

Macro BOOST_TTI_TRAIT_HAS_TEMPLATE

BOOST_TTI_TRAIT_HAS_TEMPLATE — Expands to a metafunction which tests whether an inner class template with a particular name exists.

Synopsis

```
// In header: <boost/tti/template.hpp>

BOOST_TTI_TRAIT_HAS_TEMPLATE(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner template.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' template exists within the enclosing type, otherwise 'value' is false.

The template must have all 'class' (or 'typename') parameters types.

Macro BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE

BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE — Expands to a metafunction class which tests whether an inner class template with a particular name exists.

Synopsis

```
// In header: <boost/tti/template.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner template.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' template exists within the enclosing type, otherwise 'value' is false.

The template must have all 'class' (or 'typename') parameters types.

Macro BOOST_TTI_HAS_TEMPLATE

BOOST_TTI_HAS_TEMPLATE — Expands to a metafunction which tests whether an inner class template with a particular name exists.

Synopsis

```
// In header: <boost/tti/template.hpp>

BOOST_TTI_HAS_TEMPLATE(name)
```

Description

name = the name of the inner template.

returns = a metafunction called "boost::tti::has_template_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' template exists within the enclosing type, otherwise 'value' is false.

The template must have all 'class' (or 'typename') parameters types.

Macro BOOST_TTI_MTFC_HAS_TEMPLATE

BOOST_TTI_MTFC_HAS_TEMPLATE — Expands to a metafunction class which tests whether an inner class template with a particular name exists.

Synopsis

```
// In header: <boost/tti/template.hpp>

BOOST_TTI_MTFC_HAS_TEMPLATE(name)
```

Description

name = the name of the inner template.

returns = a metafunction class called "boost::tti::mtfc_has_template_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' template exists within the enclosing type, otherwise 'value' is false.

The template must have all 'class' (or 'typename') parameters types.

Header <boost/tti/template_params.hpp>

```


BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, tpSeq)
BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, tpSeq)
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS(name, tpSeq)
BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS(name, tpSeq)
```

Macro BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/template_params.hpp>

BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, tpSeq)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner class template.

tpSeq = a Boost PP sequence which has the class template parameters. Each part of the template parameters separated by a comma (,) is put in a separate sequence element.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template with the signature as defined by the 'tpSeq' exists within the enclosing type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction class which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/template_params.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, tpSeq)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner class template.

tpSeq = a Boost PP sequence which has the class template parameters. Each part of the template parameters separated by a comma (,) is put in a separate sequence element.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template with the signature as defined by the 'tpSeq' exists within the enclosing type, otherwise 'value' is false.

Macro BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/template_params.hpp>

BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS(name, tpSeq)
```

Description

name = the name of the inner class template.

tpSeq = a Boost PP sequence which has the class template parameters. Each part of the template parameters separated by a comma (,) is put in a separate sequence element.

returns = a metafunction called "boost::tti::has_template_check_params_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template with the signature as defined by the 'tpSeq' exists within the enclosing type, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction class which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/template_params.hpp>

BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS(name, tpSeq)
```

Description

name = the name of the inner class template.

tpSeq = a Boost PP sequence which has the class template parameters. Each part of the template parameters separated by a comma (,) is put in a separate sequence element.

returns = a metafunction class called "boost::tti::mtfc_has_template_check_params_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template with the signature as defined by the 'tpSeq' exists within the enclosing type, otherwise 'value' is false.

Header <boost/tti/type.hpp>

```


BOOST_TTI_TRAIT_HAS_TYPE(trait, name)
BOOST_TTI_MTFC_TRAIT_HAS_TYPE(trait, name)
BOOST_TTI_HAS_TYPE(name)
BOOST_TTI_MTFC_HAS_TYPE(name)
```

Macro BOOST_TTI_TRAIT_HAS_TYPE

BOOST_TTI_TRAIT_HAS_TYPE — Expands to a metafunction which tests whether an inner type with a particular name exists and optionally is a particular type.

Synopsis

```
// In header: <boost/tti/type.hpp>

BOOST_TTI_TRAIT_HAS_TYPE(trait, name)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner type.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

U = the type of the inner type named 'name' as an optional parameter.

returns = 'value' is true if the 'name' type exists within the enclosing type and, if type U is specified, the 'name' type is the same as the type U, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_TRAIT_HAS_TYPE

BOOST_TTI_MTFC_TRAIT_HAS_TYPE — Expands to a metafunction class which tests whether an inner type with a particular name exists and optionally is a particular type.

Synopsis

```
// In header: <boost/tti/type.hpp>

BOOST_TTI_MTFC_TRAIT_HAS_TYPE(trait, name)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner type.

returns = a metfunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

U = the type of the inner type named 'name' as an optional parameter.

returns = 'value' is true if the 'name' type exists within the enclosing type and, if type U is specified, the 'name' type is the same as the type U, otherwise 'value' is false.

Macro BOOST_TTI_HAS_TYPE

BOOST_TTI_HAS_TYPE — Expands to a metafunction which tests whether an inner type with a particular name exists and optionally is a particular type.

Synopsis

```
// In header: <boost/tti/type.hpp>

BOOST_TTI_HAS_TYPE(name)
```

Description

name = the name of the inner type.

returns = a metafunction called "boost::tti::has_type_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

U = the type of the inner type named 'name' as an optional parameter.

returns = 'value' is true if the 'name' type exists within the enclosing type and, if type U is specified, the 'name' type is the same as the type U, otherwise 'value' is false.

Macro BOOST_TTI_MTFC_HAS_TYPE

BOOST_TTI_MTFC_HAS_TYPE — Expands to a metafunction class which tests whether an inner type with a particular name exists and optionally is a particular type.

Synopsis

```
// In header: <boost/tti/type.hpp>

BOOST_TTI_MTFC_HAS_TYPE(name)
```

Description

name = the name of the inner type.

returns = a metafunction class called "boost::tti::mtfc_has_type_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

U = the type of the inner type named 'name' as an optional parameter.

returns = 'value' is true if the 'name' type exists within the enclosing type and, if type U is specified, the 'name' type is the same as the type U, otherwise 'value' is false.

Header <boost/tti/vm_template_params.hpp>

```
BOOST_TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, ...)
BOOST_TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, ...)
BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS(name, ...)
BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS(name, ...)
```

Macro BOOST_TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/vm_template_params.hpp>

BOOST_TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, ...)
```

Description

trait = the name of the metafunction within the tti namespace.

name = the name of the inner class template.

... = variadic macro data which has the class template parameters.

returns = a metafunction called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template, with the signature as defined by the '...' variadic macro data, exists within the enclosing type, otherwise 'value' is false.

Macro BOOST_TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction class which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/vm_template_params.hpp>

BOOST_TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS(trait, name, ...)
```

Description

trait = the name of the metafunction class within the tti namespace.

name = the name of the inner class template.

... = variadic macro data which has the class template parameters.

returns = a metafunction class called "boost::tti::trait" where 'trait' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template, with the signature as defined by the '...' variadic macro data, exists within the enclosing type, otherwise 'value' is false.

Macro BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/vm_template_params.hpp>

BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS(name, ...)
```

Description

name = the name of the inner class template.

... = variadic macro data which has the class template parameters.

returns = a metafunction called "boost::tti::has_template_check_params_name" where 'name' is the macro parameter.

The metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template, with the signature as defined by the '...' variadic macro data, exists within the enclosing type, otherwise 'value' is false.

Macro BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS

BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS — Expands to a metafunction class which tests whether an inner class template with a particular name and signature exists.

Synopsis

```
// In header: <boost/tti/vm_template_params.hpp>

BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS(name, ...)
```

Description

name = the name of the inner class template.

... = variadic macro data which has the class template parameters.

returns = a metafunction class called "boost::tti::mtfc_has_template_check_params_name" where 'name' is the macro parameter.

The metafunction class's 'apply' metafunction types and return:

T = the enclosing type in which to look for our 'name'.

returns = 'value' is true if the 'name' class template, with the signature as defined by the '...' variadic macro data, exists within the enclosing type, otherwise 'value' is false.

Testing TTI

In the libs/tti/test subdirectory there is a jamfile which can be used to test TTI functionality.

Executing the jamfile without a target will run tests for both basic TTI and for the variadic macro portion of TTI. To successfully do that you need to get the `variadic_macro_data` library from the sandbox. You can run tests for only the basic TTI, which is the vast majority of TTI functionality, by specifying only the 'tti' target when executing the jamfile, and therefore you would not need the `variadic_macro_data` library. If you just want to run the tests for the variadic macro portion of TTI, specify the target as 'ttivm'.

The TTI library has been tested with VC++ 8, 9, 10 and with gcc 3.4.2, 3.4.5, 4.3.0, 4.4.0, 4.5.0-1, and 4.5.2-1.

History

Version 1.4

- Breaking changes
  - `BOOST_TTI_HAS_MEMBER` (`BOOST_TTI_TRAIT_HAS_MEMBER`) has been changed to `BOOST_TTI_HAS_COMP_MEMBER_FUNCTION` (`BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION`) and `BOOST_TTI_MTFC_HAS_MEMBER` (`BOOST_TTI_MTFC_TRAIT_HAS_MEMBER`) has been changed to `BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION` (`BOOST_TTI_MTFC_TRAIT_HAS_COMP_MEMBER_FUNCTION`). This family of functionality now supports only member functions with composite syntax.
  - `BOOST_TTI_HAS_STATIC_MEMBER` (`BOOST_TTI_TRAIT_HAS_STATIC_MEMBER`) has been changed to `BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION` (`BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION`) and `BOOST_TTI_MTFC_HAS_STATIC_MEMBER` (`BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER`) has been changed to `BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION` (`BOOST_TTI_MTFC_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION`). This family of functionality now supports only static member functions with composite syntax.

  - `boost::tti::mf_has_static_data` has been changed to `boost::tti::mf_has_static_member_data`.
- Added `BOOST_TTI_HAS_STATIC_MEMBER_DATA` and family for introspecting static member data.
- Inclusion of specific header files for faster compilation is now supported.
- Inclusion of macro metafunction name generating macros.
- Shorten the names of the test files and test header files.
- Added documentation topic about introspecting function templates.

Version 1.3

- Breaking changes
  - The names of the main header files are shortened to 'tti.hpp' and 'tti_vm.hpp'.
  - The library follows the Boost conventions.
    - Changed the filenames to lower case and underscores.
    - The top-level tti namespace has become the boost::tti namespace.
    - The macros now start with `BOOST_TTI_` rather than just `TTI_` as previously.
  - The variadic macro support works only with the latest version of the variadic_macro_library, which is version 1.3+.

Version 1.2

- Added the set of metafunction class macros for passing the macro metafunctions as metadata. This complements passing the macro metafunctions as metadata using placeholder expressions.

Version 1.1

- Library now also compiles with gcc 3.4.2 and gcc 3.4.5.
- Examples of use have been added to the documentation.
- In the documentation the previously mentioned 'nested type metafunctions' are now called "nullary type metafunctions'.
- `BOOST_TTI_HAS_TYPE` and `boost::tti::mf_has_type` now have optional typedef checking.
- New macro metafunction functionality which allows composite typed to be treated as individual types has been implemented. These include:
  - `BOOST_TTI_HAS_MEMBER_DATA`
  - `BOOST_TTI_HAS_MEMBER_FUNCTION`
  - `BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION`
- New nullary type metafunction `boost::tti::mf_has_static_member_function` uses the new underlying `BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION` macro metafunction. Its signature uses an optional MPL forward sequence for the parameter types and an optional Boost `function_types` tag type.
- New nullary type metafunctions `boost::tti::valid_member_type` and `boost::tti::mf_valid_member_type` for checking if the 'type' returned from invoking the `BOOST_TTI_MEMBER_TYPE` or `boost::tti::mf_member_type` metafunctions is valid.
- Breaking changes

- `BOOST_TTI_HAS_TYPE_CHECK_TYPEDEF` and `boost::tti::mf_has_type_check_typedef` have been removed, and the functionality in them folded into `BOOST_TTI_HAS_TYPE` and `boost::tti::mf_has_type`.
- BOOST_TTI_MEMBER_TYPE and boost::tti::mf_member_type no longer also return a 'valid' boolean constant. Use boost::tti::valid_member_type or boost::tti::mf_valid_member_type metafunctions instead (see above).
- `boost::tti::mf_has_static_function` has been removed and its functionality moved to `boost::tti::mf_has_static_member_function` (see above).
- `boost::tti::mf_member_data` uses the new underlying `BOOST_TTI_HAS_MEMBER_DATA` macro metafunction.
- The signature for `boost::tti::mf_has_member_function` has changed to use an optional MPL forward sequence for the parameter types and an optional Boost `function_types` tag type.
- All nullary type metafunctions take their corresponding macro metafunction parameter as a class in the form of a Boost MPL lambda expression instead of as a template template parameter as previously. Using a placeholder expression is the easiest way to pass the corresponding macro metafunction to its nullary type metafunction.

Version 1.0

Initial version of the library.

ToDo

- Improve tests
- Improve documentation

Acknowledgments

The TTI library came out of my effort to take the `type_traits_ext` part of the unfinished Concept Traits Library and expand it. So my first thanks go to Terje Slettebo and Tobias Schwinger, the authors of the CTL. I have taken, and hopefully improved upon, the ideas and implementation in that library, and added some new functionality.

I would also like to thank Joel Falcou for his help and his introspection work.

Two of the introspection templates are taken from the MPL and lifted into my library under a different name for the sake of completeness, so I would like to thank Aleksey Gurtovoy and David Abrahams for that library, and Daniel Walker for work on those MPL introspection macros.

Finally thanks to Anthony Williams for supplying a workaround for a Visual C++ bug which is needed for introspecting member data where the type of the member data is a compound type.

Index

BOOST_TTI_HAS_COMP_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/comp_mem_fun_gen.hpp >
BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION	Header < boost/tti/comp_static_mem_fun.hpp > Header < boost/tti/gen/comp_static_mem_fun_gen.hpp > History Macro Metafunctions Macro Metafunctions as Metadata Using the Macro Metafunctions
BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN	Header < boost/tti/gen/comp_static_mem_fun_gen.hpp >
BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/comp_static_mem_fun_gen.hpp >
BOOST_TTI_HAS_MEMBER_DATA	Header < boost/tti/gen/mem_data_gen.hpp > Header < boost/tti/mem_data.hpp > History Macro Metafunctions Macro Metafunctions as Metadata Nullary Type Metafunctions Using the Macro Metafunctions
BOOST_TTI_HAS_MEMBER_DATA_GEN	Header < boost/tti/gen/mem_data_gen.hpp >
BOOST_TTI_HAS_MEMBER_DATA_GEN_BASE	Header < boost/tti/gen/mem_data_gen.hpp >
BOOST_TTI_HAS_MEMBER_FUNCTION	Header < boost/tti/gen/mem_fun_gen.hpp > Header < boost/tti/mem_fun.hpp > History Macro Metafunctions Macro Metafunctions as Metadata Nested Types Nullary Type Metafunctions Using the Macro Metafunctions Using the Nullary Type Metafunctions
BOOST_TTI_HAS_MEMBER_FUNCTION_GEN	Header < boost/tti/gen/mem_fun_gen.hpp >
BOOST_TTI_HAS_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/mem_fun_gen.hpp >
BOOST_TTI_HAS_STATIC_MEMBER_DATA	Header < boost/tti/gen/static_mem_data_gen.hpp > Header < boost/tti/static_mem_data.hpp > History Macro Metafunctions Macro Metafunctions as Metadata Nested Types Nullary Type Metafunctions Using the Macro Metafunctions
BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN	Header < boost/tti/gen/static_mem_data_gen.hpp >

BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN_BASE	Header < boost/tti/gen/static_mem_data_gen.hpp >
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION	Header < boost/tti/gen/static_mem_fun_gen.hpp > Header < boost/tti/static_mem_fun.hpp > History Macro Metafunctions Macro Metafunctions as Metadata Nested Types Nullary Type Metafunctions Using the Macro Metafunctions Using the Nullary Type Metafunctions
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN	Header < boost/tti/gen/static_mem_fun_gen.hpp >
BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/static_mem_fun_gen.hpp >
BOOST_TTI_HAS_TEMPLATE	Header < boost/tti/gen/template_gen.hpp > Header < boost/tti/template.hpp > Introspecting Function Templates Macro Metafunctions Macro Metafunctions as Metadata Nullary Type Metafunctions Using the Macro Metafunctions
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS	Header < boost/tti/gen/template_params_gen.hpp > Header < boost/tti/template_params.hpp > Introspecting Function Templates Macro Metafunctions Macro Metafunctions as Metadata Nullary Type Metafunctions Using the Macro Metafunctions Using the Nullary Type Metafunctions
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN	Header < boost/tti/gen/template_params_gen.hpp >
BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE	Header < boost/tti/gen/template_params_gen.hpp >
BOOST_TTI_HAS_TEMPLATE_GEN	Header < boost/tti/gen/template_gen.hpp >
BOOST_TTI_HAS_TEMPLATE_GEN_BASE	Header < boost/tti/gen/template_gen.hpp >
BOOST_TTI_HAS_TYPE	General Functionality Header < boost/tti/gen/type_gen.hpp > Header < boost/tti/type.hpp > History Macro Metafunctions Macro Metafunctions as Metadata Nested Types Nullary Type Metafunctions Using the Macro Metafunctions Using the Nullary Type Metafunctions

BOOST_TTI_HAS_TYPE_GEN	Header < boost/tti/gen/type_gen.hpp >
BOOST_TTI_HAS_TYPE_GEN_BASE	Header < boost/tti/gen/type_gen.hpp >
BOOST_TTI_MEMBER_TYPE	Header < boost/tti/gen/mem_type_gen.hpp > Header < boost/tti/mem_type.hpp > History Introspecting Function Templates Macro Metafunctions Macro Metafunctions as Metadata Nested Types Nullary Type Metafunctions Using the Macro Metafunctions Using the Nullary Type Metafunctions
BOOST_TTI_MEMBER_TYPE_GEN	Header < boost/tti/gen/mem_type_gen.hpp >
BOOST_TTI_MEMBER_TYPE_GEN_BASE	Header < boost/tti/gen/mem_type_gen.hpp >
BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION	Header < boost/tti/comp_mem_fun.hpp > Header < boost/tti/gen/comp_mem_fun_gen.hpp > History Macro Metafunctions as Metadata
BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN	Header < boost/tti/gen/comp_mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_COMP_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/comp_mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION	Header < boost/tti/comp_static_mem_fun.hpp > Header < boost/tti/gen/comp_static_mem_fun_gen.hpp > History Macro Metafunctions as Metadata
BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN	Header < boost/tti/gen/comp_static_mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_COMP_STATIC_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/comp_static_mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_MEMBER_DATA	Header < boost/tti/gen/mem_data_gen.hpp > Header < boost/tti/mem_data.hpp > Macro Metafunctions as Metadata Using the Nullary Type Metafunctions
BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN	Header < boost/tti/gen/mem_data_gen.hpp >
BOOST_TTI_MTFC_HAS_MEMBER_DATA_GEN_BASE	Header < boost/tti/gen/mem_data_gen.hpp >
BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION	Header < boost/tti/gen/mem_fun_gen.hpp > Header < boost/tti/mem_fun.hpp >

	Macro Metafunctions as Metadata
BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN	Header < boost/tti/gen/mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA	Header < boost/tti/gen/static_mem_data_gen.hpp > Header < boost/tti/static_mem_data.hpp > Macro Metafunctions as Metadata Using the Nullary Type Metafunctions
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN	Header < boost/tti/gen/static_mem_data_gen.hpp >
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN_BASE	Header < boost/tti/gen/static_mem_data_gen.hpp >
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION	Header < boost/tti/gen/static_mem_fun_gen.hpp > Header < boost/tti/static_mem_fun.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN	Header < boost/tti/gen/static_mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE	Header < boost/tti/gen/static_mem_fun_gen.hpp >
BOOST_TTI_MTFC_HAS_TEMPLATE	Header < boost/tti/gen/template_gen.hpp > Header < boost/tti/template.hpp > Macro Metafunctions as Metadata Using the Nullary Type Metafunctions
BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS	Header < boost/tti/gen/template_params_gen.hpp > Header < boost/tti/template_params.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN	Header < boost/tti/gen/template_params_gen.hpp >
BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE	Header < boost/tti/gen/template_params_gen.hpp >
BOOST_TTI_MTFC_HAS_TEMPLATE_GEN	Header < boost/tti/gen/template_gen.hpp >
BOOST_TTI_MTFC_HAS_TEMPLATE_GEN_BASE	Header < boost/tti/gen/template_gen.hpp >
BOOST_TTI_MTFC_HAS_TYPE	Header < boost/tti/gen/type_gen.hpp > Header < boost/tti/type.hpp > Macro Metafunctions as Metadata Using the Nullary Type Metafunctions
BOOST_TTI_MTFC_HAS_TYPE_GEN	Header < boost/tti/gen/type_gen.hpp >

BOOST_TTI_MTFC_HAS_TYPE_GEN_BASE	Header < boost/tti/gen/type_gen.hpp >
BOOST_TTI_MTFC_MEMBER_TYPE	Header < boost/tti/gen/mem_type_gen.hpp > Header < boost/tti/mem_type.hpp > Macro Metafunctions as Metadata Using the Nullary Type Metafunctions
BOOST_TTI_MTFC_MEMBER_TYPE_GEN	Header < boost/tti/gen/mem_type_gen.hpp >
BOOST_TTI_MTFC_MEMBER_TYPE_GEN_BASE	Header < boost/tti/gen/mem_type_gen.hpp >
BOOST_TTI_MTFC_TRAIT_HAS_COMP_MEMBER_FUNCTION	Header < boost/tti/comp_mem_fun.hpp > History Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION	Header < boost/tti/comp_static_mem_fun.hpp > History Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_DATA	Header < boost/tti/mem_data.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION	Header < boost/tti/mem_fun.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA	Header < boost/tti/static_mem_data.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION	Header < boost/tti/static_mem_fun.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE	Header < boost/tti/template.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS	Header < boost/tti/template_params.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_HAS_TYPE	Header < boost/tti/type.hpp > Macro Metafunctions as Metadata
BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE	Header < boost/tti/mem_type.hpp > Macro Metafunctions as Metadata
BOOST_TTI_NAMESPACE	General Functionality Header < boost/tti/gen/namespace_gen.hpp > Header < boost/tti/mf/mf_type.hpp >
BOOST_TTI_TRAIT_GEN	General Functionality Header < boost/tti/gen/trait_gen.hpp >
BOOST_TTI_TRAIT_GEN_BASE	General Functionality Header < boost/tti/gen/trait_gen.hpp >

BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION	Header < boost/tti/comp_mem_fun.hpp > History Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION	Header < boost/tti/comp_static_mem_fun.hpp > History Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_MEMBER_DATA	Header < boost/tti/mem_data.hpp > Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION	Header < boost/tti/mem_fun.hpp > Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA	Header < boost/tti/static_mem_data.hpp > Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION	Header < boost/tti/static_mem_fun.hpp > Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_TEMPLATE	Header < boost/tti/template.hpp > Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS	Header < boost/tti/template_params.hpp > Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_HAS_TYPE	General Functionality Header < boost/tti/type.hpp > Macro Metafunctions as Metadata
BOOST_TTI_TRAIT_MEMBER_TYPE	Header < boost/tti/mem_type.hpp > Macro Metafunctions as Metadata
BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS	Header < boost/tti/gen/vm_template_params_gen.hpp > Header < boost/tti/vm_template_params.hpp > Macro Metafunctions Macro Metafunctions as Metadata Nullary Type Metafunctions Using the Macro Metafunctions
BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN	Header < boost/tti/gen/vm_template_params_gen.hpp >
BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE	Header < boost/tti/gen/vm_template_params_gen.hpp >
BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS	Header < boost/tti/gen/vm_template_params_gen.hpp > Header < boost/tti/vm_template_params.hpp > Macro Metafunctions as Metadata Using the Nullary Type Metafunctions
BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN	Header < boost/tti/gen/vm_template_params_gen.hpp >
BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE	Header < boost/tti/gen/vm_template_params_gen.hpp >

Header < boost/tti/gen/namespace_gen.hpp >	BOOST_TTI_NAMESPACE
Header < boost/tti/gen/static_mem_data_gen.hpp >	BOOST_TTI_HAS_STATIC_MEMBER_DATA BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN BOOST_TTI_HAS_STATIC_MEMBER_DATA_GEN_BASE BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA_GEN_BASE
Header < boost/tti/gen/static_mem_fun_gen.hpp >	BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION_GEN_BASE
Header < boost/tti/gen/template_gen.hpp >	BOOST_TTI_HAS_TEMPLATE BOOST_TTI_HAS_TEMPLATE_GEN BOOST_TTI_HAS_TEMPLATE_GEN_BASE BOOST_TTI_MTFC_HAS_TEMPLATE BOOST_TTI_MTFC_HAS_TEMPLATE_GEN BOOST_TTI_MTFC_HAS_TEMPLATE_GEN_BASE
Header < boost/tti/gen/template_params_gen.hpp >	BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE
Header < boost/tti/gen/trait_gen.hpp >	BOOST_TTI_TRAIT_GEN BOOST_TTI_TRAIT_GEN_BASE
Header < boost/tti/gen/type_gen.hpp >	BOOST_TTI_HAS_TYPE BOOST_TTI_HAS_TYPE_GEN BOOST_TTI_HAS_TYPE_GEN_BASE BOOST_TTI_MTFC_HAS_TYPE BOOST_TTI_MTFC_HAS_TYPE_GEN BOOST_TTI_MTFC_HAS_TYPE_GEN_BASE
Header < boost/tti/gen/vm_template_params_gen.hpp >	BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS_GEN_BASE
Header < boost/tti/mem_data.hpp >	BOOST_TTI_HAS_MEMBER_DATA BOOST_TTI_MTFC_HAS_MEMBER_DATA BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_DATA BOOST_TTI_TRAIT_HAS_MEMBER_DATA
Header < boost/tti/mem_fun.hpp >	BOOST_TTI_HAS_MEMBER_FUNCTION BOOST_TTI_MTFC_HAS_MEMBER_FUNCTION BOOST_TTI_MTFC_TRAIT_HAS_MEMBER_FUNCTION BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION
Header < boost/tti/mem_type.hpp >	BOOST_TTI_MEMBER_TYPE

	BOOST_TTI_MTFC_MEMBER_TYPE BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE BOOST_TTI_TRAIT_MEMBER_TYPE
Header < boost/tti/mf/mf_mem_data.hpp >	mf_has_member_data
Header < boost/tti/mf/mf_mem_fun.hpp >	mf_has_member_function
Header < boost/tti/mf/mf_mem_type.hpp >	mf_member_type mf_valid_member_type valid_member_type
Header < boost/tti/mf/mf_static_mem_data.hpp >	mf_has_static_member_data
Header < boost/tti/mf/mf_static_mem_fun.hpp >	mf_has_static_member_function
Header < boost/tti/mf/mf_template.hpp >	mf_has_template
Header < boost/tti/mf/mf_template_params.hpp >	mf_has_template_check_params
Header < boost/tti/mf/mf_type.hpp >	BOOST_TTI_NAMESPACE mf_has_type
Header < boost/tti/static_mem_data.hpp >	BOOST_TTI_HAS_STATIC_MEMBER_DATA BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_DATA BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA
Header < boost/tti/static_mem_fun.hpp >	BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_MTFC_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_MTFC_TRAIT_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION
Header < boost/tti/template.hpp >	BOOST_TTI_HAS_TEMPLATE BOOST_TTI_MTFC_HAS_TEMPLATE BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE BOOST_TTI_TRAIT_HAS_TEMPLATE
Header < boost/tti/template_params.hpp >	BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_MTFC_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS
Header < boost/tti/type.hpp >	BOOST_TTI_HAS_TYPE BOOST_TTI_MTFC_HAS_TYPE BOOST_TTI_MTFC_TRAIT_HAS_TYPE BOOST_TTI_TRAIT_HAS_TYPE
Header < boost/tti/vm_template_params.hpp >	BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS
History	BOOST_TTI_HAS_COMP_MEMBER_FUNCTION

		BOOST_TTI_MTFC_TRAIT_HAS_TYPE BOOST_TTI_MTFC_TRAIT_MEMBER_TYPE BOOST_TTI_TRAIT_HAS_COMP_MEMBER_FUNCTION BOOST_TTI_TRAIT_HAS_COMP_STATIC_MEMBER_FUNCTION BOOST_TTI_TRAIT_HAS_MEMBER_DATA BOOST_TTI_TRAIT_HAS_MEMBER_FUNCTION BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_DATA BOOST_TTI_TRAIT_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_TRAIT_HAS_TEMPLATE BOOST_TTI_TRAIT_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_TRAIT_HAS_TYPE BOOST_TTI_TRAIT_MEMBER_TYPE BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_MTFC_TRAIT_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_VM_TRAIT_HAS_TEMPLATE_CHECK_PARAMS
	mf_has_member_data	Header < boost/tti/mf/mf_mem_data.hpp >
	mf_has_member_function	Header < boost/tti/mf/mf_mem_fun.hpp >
	mf_has_static_member_data	Header < boost/tti/mf/mf_static_mem_data.hpp >
	mf_has_static_member_function	Header < boost/tti/mf/mf_static_mem_fun.hpp >
	mf_has_template	Header < boost/tti/mf/mf_template.hpp >
	mf_has_template_check_params	Header < boost/tti/mf/mf_template_params.hpp >
	mf_has_type	Header < boost/tti/mf/mf_type.hpp >
	mf_member_type	Header < boost/tti/mf/mf_mem_type.hpp >
	mf_valid_member_type	Header < boost/tti/mf/mf_mem_type.hpp >
N	Nested Types	BOOST_TTI_HAS_MEMBER_FUNCTION BOOST_TTI_HAS_STATIC_MEMBER_DATA BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_HAS_TYPE BOOST_TTI_MEMBER_TYPE
	Nullary Type Metafunctions	BOOST_TTI_HAS_MEMBER_DATA BOOST_TTI_HAS_MEMBER_FUNCTION BOOST_TTI_HAS_STATIC_MEMBER_DATA BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_HAS_TEMPLATE BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_HAS_TYPE BOOST_TTI_MEMBER_TYPE BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS
U	Using the Macro Metafunctions	BOOST_TTI_HAS_COMP_MEMBER_FUNCTION BOOST_TTI_HAS_COMP_STATIC_MEMBER_FUNCTION BOOST_TTI_HAS_MEMBER_DATA BOOST_TTI_HAS_MEMBER_FUNCTION BOOST_TTI_HAS_STATIC_MEMBER_DATA BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_HAS_TEMPLATE BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_HAS_TYPE

		BOOST_TTI_MEMBER_TYPE BOOST_TTI_VM_HAS_TEMPLATE_CHECK_PARAMS
	Using the Nullary Type Metafunctions	BOOST_TTI_HAS_MEMBER_FUNCTION BOOST_TTI_HAS_STATIC_MEMBER_FUNCTION BOOST_TTI_HAS_TEMPLATE_CHECK_PARAMS BOOST_TTI_HAS_TYPE BOOST_TTI_MEMBER_TYPE BOOST_TTI_MTFC_HAS_MEMBER_DATA BOOST_TTI_MTFC_HAS_STATIC_MEMBER_DATA BOOST_TTI_MTFC_HAS_TEMPLATE BOOST_TTI_MTFC_HAS_TYPE BOOST_TTI_MTFC_MEMBER_TYPE BOOST_TTI_VM_MTFC_HAS_TEMPLATE_CHECK_PARAMS
V	valid_member_type	Header < boost/tti/mf/mf_mem_type.hpp >