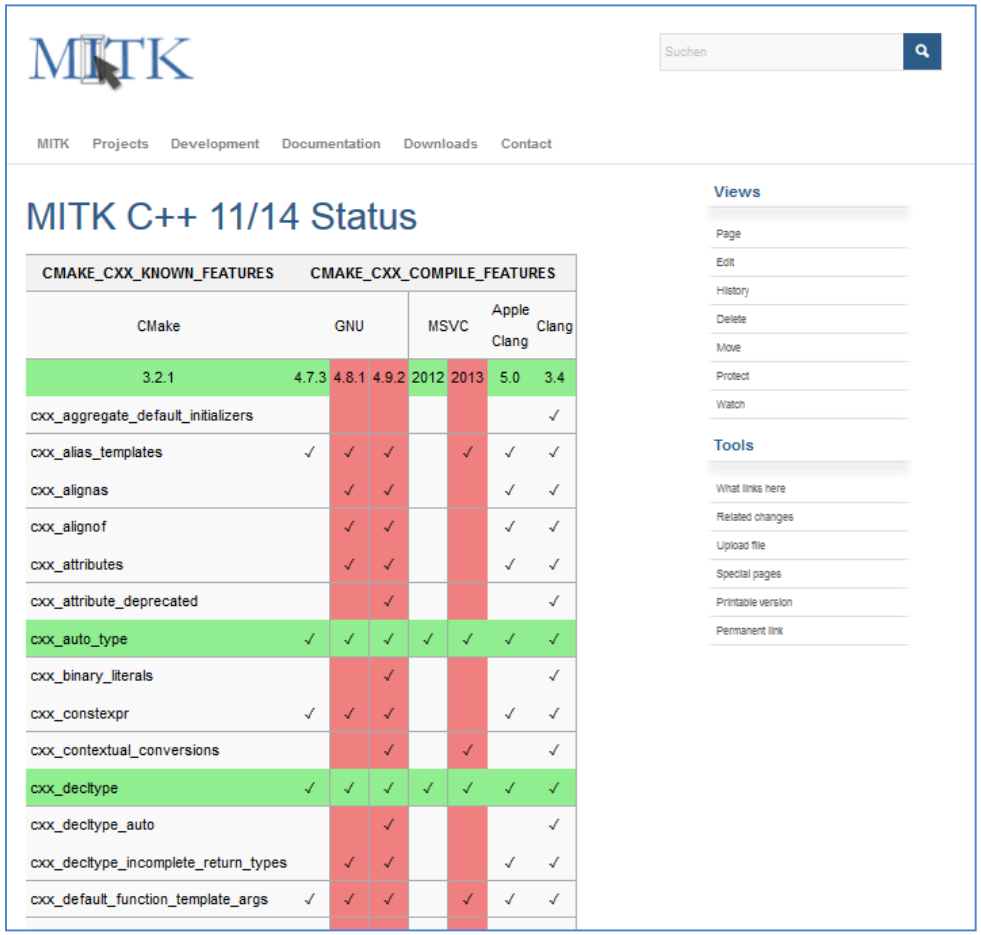
C++ 11

ENUM FORWARD DECLARATIONS & STRONG ENUMS

MITK C++ 11/14 STATUS



MITK

Suchen

MITK Projects Development Documentation Downloads Contact

MITK C++ 11/14 Status

CMAKE_CXX_KNOWN_FEATURES	CMAKE_CXX_COMPILE_FEATURES						
CMake	GNU			MSVC		Apple Clang	Clang
3.2.1	4.7.3	4.8.1	4.9.2	2012	2013	5.0	3.4
cxx_aggregate_default_initializers							✓
cxx_alias_templates	✓	✓	✓		✓	✓	✓
cxx_alignas		✓	✓			✓	✓
cxx_alignof		✓	✓			✓	✓
cxx_attributes		✓	✓			✓	✓
cxx_attribute_deprecated			✓				✓
cxx_auto_type	✓	✓	✓	✓	✓	✓	✓
cxx_binary_literals			✓				✓
cxx_constexpr	✓	✓	✓			✓	✓
cxx_contextual_conversions			✓		✓		✓
cxx_decltype	✓	✓	✓	✓	✓	✓	✓
cxx_decltype_auto			✓				✓
cxx_decltype_incomplete_return_types		✓	✓			✓	✓
cxx_default_function_template_args	✓	✓	✓		✓	✓	✓

Views

Page
Edit
History
Delete
Move
Protect
Watch

Tools

What links here
Related changes
Upload file
Special pages
Printable version
Permanent link

http://mitk.org/wiki/MITK_C++_11/14_Status

PROBLEMS WITH TRADITIONAL ENUMS

- **name clashes**, if two enumerators in different enums declared in the same scope have the same name;

```
enum Animals {Bear, Cat, Chicken};
enum Birds {Eagle, Duck, Chicken}; // error! Chicken has already been declared!
```

- not possible to use an enumerator with a **fully qualified name**

```
Animals animal = Animals::Cat; //error or warning Animals animal = Cat; // Ok!
```

- enumerations are **not type-safe**

UNSCOPED ENUMERATION

```
enum name { enumerator = constexpr , enumerator = constexpr ,
                ... }
```

- declares an unscoped enumeration type whose **underlying type** is **not fixed**

Since C++ 11:

```
enum name : type { enumerator = constexpr , enumerator =
                        constexpr , ... }
```

- declares an unscoped enumeration type whose **underlying type** is **fixed**

EXAMPLE
TRADITIONAL ENUMS

```
enum Color { red, yellow, green=20, blue };
Color col = red;
int n = blue; // n == 21

struct Foo {
 enum Direction { left='l', right='r' };
};

Foo x;
int a = Foo::left;
int b = x.left;
int c = Foo::Direction::left; // allowed only in C++11 and
later
```

SCOPED ENUMERATIONS

```
enum struct|class name { enumerator = constexpr , enumerator
= constexpr , ... }
```

- declares a scoped enumeration type whose **underlying type** is **int**

```
enum struct|class name : type { enumerator = constexpr ,
enumerator = constexpr , ... }
```

- declares a scoped enumeration type whose **underlying type** is **type**

EXAMPLE
IMPLICIT CONVERSION

```
enum class Color { RED, GREEN=20, BLUE };
Color r = Color::BLUE;
switch(r) {
  case Color::RED : std::cout << "red" "\n";
  break;
  case Color::GREEN : std::cout << "green" "\n";
  break;
  case Color::BLUE : std::cout << "blue" "\n";
  break;
}

int n = r; // error: no scoped enum to int conversion
int n = static_cast<int>(r); // OK, n = 21
```

EXAMPLE
ENUM FORWARD DECLARATION

- underlying type can be specified, so forward declaration is possible

```
enum class Selection : unsigned char;

void make_selection(Selection s) { }

enum class Selection : unsigned char
{
None,
Single,
Multiple,
};
```

CONCLUSION

What‘s new:

- New type of enums: strongly-typed enums
- Solve the known problems with the traditional enums
  - the scope of the enumerators
  - possibility to specify the underlying type (also traditional)
  - type-safe
  - Forward-declaring

REFERENCES

- http://mitk.org/wiki/MITK_C++_11/14_Status
- http://en.cppreference.com/w/cpp/language/enum
- http://www.cprogramming.com/tutorial/enum.html
- http://stackoverflow.com/questions/12581064/enum-vs-strongly-typed-enum