RPN

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 RPN Namespace Reference

Classes

- struct EquationValidator
- struct NotationConverter
- struct NotationDeterminer
- struct RPNSolver
- struct Spacer
- struct TokenReader

Functions

- int sumLetters (const std::string &str)
- double handleDivision (const double &a, const double &b)
- double handleSqrt (const double &a)
- double handleCbrt (const double &a)
- double calculate (const double &a, const double &b, const std::string &op)
- double calculate (const double &a, const std::string &op)
- bool isOperator (const std::string &op)
- bool is1ArgOperator (const std::string &op)
- bool is2ArgOperator (const std::string &op)

Variables

- constexpr int EXP_PREC = 100
- constexpr int TRIG_FUN_PREC = EXP_PREC-1
- constexpr int MULT_DIV_PREC = TRIG_FUN_PREC-1
- constexpr int ADD_SUB_PREC = MULT_DIV_PREC-1
- const std::map< std::string, int > operatorPrecedence
- const std::unordered_set< std::string > one_arg_operators
- const std::unordered_set< std::string > two_arg_operators

4.1.1 Function Documentation

4.1.1.1 calculate() [1/2]

```
double RPN::calculate (
            const double & a,
            const double & b,
            const std::string & op)
```

Given operator and operands calculates the result

Parameters

a	left operand
b	right operand
op	operator

Returns

result

Definition at line 76 of file RPN.cpp.

4.1.1.2 calculate() [2/2]

```
double RPN::calculate (
            const double & a,
            const std::string & op)
```

Given operator and operand calculates the result

Parameters

a	operand
op	operator

Returns

result

Integers found in cases of this switch come from the sum of ascii values of letters of the operators.

Definition at line 101 of file RPN.cpp.

4.1.1.3 handleCbrt()

```
double RPN::handleCbrt (
            const double & a)
```

Calculates cubic roots and errors on negative numbers.

Parameters

a	

Returns

cbrt(a)

Definition at line 61 of file RPN.cpp.

4.1.1.4 handleDivision()

```
double RPN::handleDivision (
            const double & a,
            const double & b)
```

Handler for division. Throws error on divisor = 0.

Parameters

a	
b	

Returns

a/b

Definition at line 35 of file RPN.cpp.

4.1.1.5 handleSqrt()

```
double RPN::handleSqrt (
            const double & a)
```

Calculates square roots and errors on negative numbers.

Parameters

a	

Returns

sqrt(a)

Definition at line 48 of file RPN.cpp.

4.1.1.6 is1ArgOperator()

```
bool RPN::is1ArgOperator (
            const std::string & op)
```

Checks if given token is an operator that takes only 1 argument, e.g. sqrt(x).

Returns

true if is 1 argument operator.

Definition at line 195 of file RPN.cpp.

4.1.1.7 is2ArgOperator()

```
bool RPN::is2ArgOperator (
            const std::string & op)
```

Checks if given token is an operator that takes 2 arguments, e.g. a + b.

Returns

true if is 2 argument operator.

Definition at line 204 of file RPN.cpp.

4.1.1.8 isOperator()

```
bool RPN::isOperator (
            const std::string & op)
```

Checks if given string is a valid operator

Parameters

op	

Returns

true if string is an operator

Definition at line 186 of file RPN.cpp.

4.1.1.9 sumLetters()

```
int RPN::sumLetters (
            const std::string & str)
```

Intermediate function used by calculate for 1 parameter operators. Sums ascii values of letters to determine which switch case use.

Parameters

str	

Returns

Ascii sum of letters.

Definition at line 21 of file RPN.cpp.

4.1.2 Variable Documentation

4.1.2.1 ADD_SUB_PREC

```
int RPN::ADD_SUB_PREC = MULT_DIV_PREC-1  [constexpr]
```

Addition/subtraction precedence score.

Definition at line 139 of file RPN.cpp.

4.1.2.2 EXP_PREC

```
int RPN::EXP_PREC = 100  [constexpr]
```

Exponential precedence score.

Definition at line 127 of file RPN.cpp.

4.1.2.3 MULT_DIV_PREC

```
int RPN::MULT_DIV_PREC = TRIG_FUN_PREC-1  [constexpr]
```

Multiplication/division precedence score.

Definition at line 135 of file RPN.cpp.

4.1.2.4 one_arg_operators

```
const std::unordered_set<std::string> RPN::one_arg_operators
```

Initial value:

```
= {
        "sqrt",
        "cbrt",
        "sin",
        "cos",
        "tan",
    }
```

Operators taking 1 parameter

Definition at line 161 of file RPN.cpp.

4.1.2.5 operatorPrecedence

```
const std::map<std::string, int> RPN::operatorPrecedence
```

Initial value:

```
= {
        {"^", EXP_PREC},
        {"sqrt", EXP_PREC},
        {"cbrt", EXP_PREC},
        {"sin", TRIG_FUN_PREC},
        {"cos", TRIG_FUN_PREC},
        {"tan", TRIG_FUN_PREC},
        {"*", MULT_DIV_PREC},
        {"/", MULT_DIV_PREC},
        {"\\", MULT_DIV_PREC},
        {"+", ADD_SUB_PREC},
        {"-", ADD_SUB_PREC},
    }
```

Mapped precedences to operators.

Definition at line 144 of file RPN.cpp.

4.1.2.6 TRIG_FUN_PREC

```
int RPN::TRIG_FUN_PREC = EXP_PREC-1  [constexpr]
```

Trigonometric functions precedence score.

Definition at line 131 of file RPN.cpp.

4.1.2.7 two_arg_operators

```
const std::unordered_set<std::string> RPN::two_arg_operators
```

Initial value:

```
= {
        "^",
        "*",
        "/",
        "\\",
        "+",
        "-"
    }
```

Operators taking 2 parameters

Definition at line 172 of file RPN.cpp.

Chapter 5

Class Documentation

5.1 RPN::EquationValidator Struct Reference

`#include <RPN.h>`

Static Public Member Functions

- static bool isValidRPN (const std::string &equation)
- static bool isValidInfix (const std::string &equation)

5.1.1 Detailed Description

Definition at line 133 of file RPN.h.

5.1.2 Member Function Documentation

5.1.2.1 isValidInfix()

```
bool RPN::EquationValidator::isValidInfix (
            const std::string & equation)  [static]
```

Validates Infix equation.

Parameters

equation	Infix equation

Definition at line 491 of file RPN.cpp.

5.1.2.2 isValidRPN()

```
bool RPN::EquationValidator::isValidRPN (
            const std::string & equation)  [static]
```

Validates RPN equation.

Parameters

equation	RPN equation

Definition at line 457 of file RPN.cpp.

The documentation for this struct was generated from the following files:

- lib/RPN.h
- lib/RPN.cpp

5.2 RPN::NotationConverter Struct Reference

`#include <RPN.h>`

Static Public Member Functions

- static std::string infixToRPN (const std::string &infix)
- static std::string RPNtoInfix (const std::string &RPN)

5.2.1 Detailed Description

Struct able to convert Infix to RPN and vice versa.

Definition at line 56 of file RPN.h.

5.2.2 Member Function Documentation

5.2.2.1 infixToRPN()

```
std::string RPN::NotationConverter::infixToRPN (
            const std::string & infix)  [static]
```

Given infix equation string, converts it into RPN equation.

Parameters

infix	Infix equation

Returns

RPN equation.

Definition at line 290 of file RPN.cpp.

5.2.2.2 RPNtoInfix()

```
std::string RPN::NotationConverter::RPNtoInfix (
            const std::string & RPN)  [static]
```

Given RPN equation string, converts it into infix equation.

Parameters

RPN	RPN equation.

Returns

Infix equation.

Definition at line 331 of file RPN.cpp.

The documentation for this struct was generated from the following files:

- lib/RPN.h
- lib/RPN.cpp

5.3 RPN::NotationDeterminer Struct Reference

```
#include <RPN.h>
```

Static Public Member Functions

- static bool isRPN (const std::string &equation)
- static bool isInfix (const std::string &equation)

5.3.1 Detailed Description

Definition at line 91 of file RPN.h.

5.3.2 Member Function Documentation

5.3.2.1 isInfix()

```
bool RPN::NotationDeterminer::isInfix (
            const std::string & equation)  [static]
```

Determines if string is an Infix equation.

Returns

true if equation is written in Infix.

Definition at line 377 of file RPN.cpp.

5.3.2.2 isRPN()

```
bool RPN::NotationDeterminer::isRPN (
            const std::string & equation)  [static]
```

Determines if string is an RPN equation.

Returns

true if equation is written in RPN.

Definition at line 367 of file RPN.cpp.

The documentation for this struct was generated from the following files:

- lib/RPN.h
- lib/RPN.cpp

5.4 RPN::RPNSolver Struct Reference

```
#include <RPN.h>
```

Static Public Member Functions

- static double getResult (const std::string &equation)

5.4.1 Detailed Description

RPN equation solver.

Definition at line 45 of file RPN.h.

5.4.2 Member Function Documentation

5.4.2.1 getResult()

```
double RPN::RPNSolver::getResult (
            const std::string & equation)  [static]
```

Solves for the result of the RPN equation.

Returns

Result of the RPN equation.

Takes 2 tokens from the stack, removing the first and reassigning the second to the result of the operation.

After the entire algorithm is done the stack should contain only 1 token, which is equal to the result of the equation.

Definition at line 208 of file RPN.cpp.

The documentation for this struct was generated from the following files:

- lib/RPN.h
- lib/RPN.cpp

5.5 RPN::Spacer Struct Reference

`#include <RPN.h>`

Static Public Member Functions

- static std::string addSpacesAroundParentheses (const std::string &input)
- static std::string removeSpacesAroundParentheses (const std::string &input)
- static std::string addSpacesAroundOperators (const std::string &input)
- static std::string removeSpacesAroundOperators (const std::string &input)
- static std::string mergeSpaces (const std::string &input)

5.5.1 Detailed Description

Definition at line 104 of file RPN.h.

5.5.2 Member Function Documentation

5.5.2.1 addSpacesAroundOperators()

```
std::string RPN::Spacer::addSpacesAroundOperators (
            const std::string & input)  [static]
```

Adds spaces around each operator in the equation.

Definition at line 407 of file RPN.cpp.

5.5.2.2 addSpacesAroundParentheses()

```
std::string RPN::Spacer::addSpacesAroundParentheses (
            const std::string & input)  [static]
```

Prepares equation to be fed into NotationConverter. Key idea is that each token in the converter needs to be separated by a space, that is operands, operators, parentheses.

Definition at line 381 of file RPN.cpp.

5.5.2.3 mergeSpaces()

```
std::string RPN::Spacer::mergeSpaces (
            const std::string & input)  [static]
```

Combines multiple spaces into single space character.

Definition at line 446 of file RPN.cpp.

5.5.2.4 removeSpacesAroundOperators()

```
std::string RPN::Spacer::removeSpacesAroundOperators (
            const std::string & input)  [static]
```

Removes spaces around each operator in the equation.

Definition at line 426 of file RPN.cpp.

5.5.2.5 removeSpacesAroundParentheses()

```
std::string RPN::Spacer::removeSpacesAroundParentheses (
            const std::string & input)  [static]
```

Removes all spaces around parentheses.

Definition at line 395 of file RPN.cpp.

The documentation for this struct was generated from the following files:

- lib/RPN.h
- lib/RPN.cpp

5.6 RPN::TokenReader Struct Reference

```
#include <RPN.h>
```

Public Member Functions

- TokenReader (const std::string &string)
- std::string getString ()
- std::string next ()
- std::string peek ()
- bool finished () const

5.6.1 Detailed Description

Wrapper over std::stringstream for extracting tokens from the string.

Definition at line 10 of file RPN.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 TokenReader()

```
RPN::TokenReader::TokenReader (
            const std::string & string)  [explicit]
```

Token reader constructor

Parameters

string	Reference to the string from which tokens are read.

Definition at line 240 of file RPN.cpp.

5.6.3 Member Function Documentation

5.6.3.1 finished()

```
bool RPN::TokenReader::finished () const
```

Checks if stream came to an end.

Returns

true if stream has finished.

Definition at line 255 of file RPN.cpp.

5.6.3.2 getString()

```
std::string RPN::TokenReader::getString ()
```

Returns the entire string from which reader reads.

Returns

Whole string

Definition at line 251 of file RPN.cpp.

5.6.3.3 next()

```
std::string RPN::TokenReader::next ()
```

Next tokens in the stream.

Returns

Next token.

Definition at line 245 of file RPN.cpp.

5.6.3.4 peek()

```
std::string RPN::TokenReader::peek ()
```

Checks upcoming token in the stream, while keeping the current position in the stream.

Returns

Upcoming token in the stream.

Definition at line 259 of file RPN.cpp.

The documentation for this struct was generated from the following files:

- lib/RPN.h
- lib/RPN.cpp

Chapter 6

File Documentation

6.1 build/CMakeFiles/3.30.5/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x)
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X)
- #define STRINGIFY(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_STD_99 199901L
- #define C_STD_11 201112L
- #define C_STD_17 201710L
- #define C_STD_23 202311L
- #define C_VERSION

Functions

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

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

6.1.1 Macro Definition Documentation

6.1.1.1 __has_include

```
#define __has_include(
              x)
```

Value:

```
0
```

Definition at line 17 of file CMakeCCompilerId.c.

6.1.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 745 of file CMakeCCompilerId.c.

6.1.1.3 C_STD_11

```
#define C_STD_11 201112L
```

Definition at line 831 of file CMakeCCompilerId.c.

6.1.1.4 C_STD_17

```
#define C_STD_17 201710L
```

Definition at line 832 of file CMakeCCompilerId.c.

6.1.1.5 C_STD_23

```
#define C_STD_23 202311L
```

Definition at line 833 of file CMakeCCompilerId.c.

6.1.1.6 C_STD_99

```
#define C_STD_99 199901L
```

Definition at line 830 of file CMakeCCompilerId.c.

6.1.1.7 C_VERSION

```
#define C_VERSION
```

Definition at line 843 of file CMakeCCompilerId.c.

6.1.1.8 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 448 of file CMakeCCompilerId.c.

6.1.1.9 DEC

```
#define DEC(
               n)
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 749 of file CMakeCCompilerId.c.

6.1.1.10 HEX

```
#define HEX(
               n)
```

Value:

```
  ('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 760 of file CMakeCCompilerId.c.

6.1.1.11 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 579 of file CMakeCCompilerId.c.

6.1.1.12 STRINGIFY

```
#define STRINGIFY(
               X)
```

Value:

```
STRINGIFY_HELPER(X)
```

Definition at line 469 of file CMakeCCompilerId.c.

6.1.1.13 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X)
```

Value:

```
#X
```

Definition at line 468 of file CMakeCCompilerId.c.

6.1.2 Function Documentation

6.1.2.1 main()

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

Definition at line 877 of file CMakeCCompilerId.c.

6.1.3 Variable Documentation

6.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 826 of file CMakeCCompilerId.c.

6.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 455 of file CMakeCCompilerId.c.

6.1.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["



  "OFF"
"]"
```

Definition at line 859 of file CMakeCCompilerId.c.

6.1.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 856 of file CMakeCCompilerId.c.

6.1.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 825 of file CMakeCCompilerId.c.

6.2 CMakeCCompilerId.c

Go to the documentation of this file.

```
00001 #ifdef __cplusplus
00002 # error "A C++ compiler has been selected for C."
00003 #endif
00004
00005 #if defined(__18CXX)
00006 # define ID_VOID_MAIN
00007 #endif
00008 #if defined(__CLASSIC_C__)
00009 /* cv-qualifiers did not exist in K&R C */
00010 # define const
00011 # define volatile
00012 #endif
00013
00014 #if !defined(__has_include)
00015 /* If the compiler does not have __has_include, pretend the answer is
00016    always no.  */
00017 #  define __has_include(x) 0
00018 #endif
00019
00020
00021 /* Version number components: V=Version, R=Revision, P=Patch
00022    Version date components:   YYYY=Year, MM=Month,   DD=Day  */
00023
00024 #if defined(__INTEL_COMPILER) || defined(__ICC)
00025 # define COMPILER_ID "Intel"
00026 # if defined(_MSC_VER)
00027 #  define SIMULATE_ID "MSVC"
00028 # endif
00029 # if defined(__GNUC__)
00030 #  define SIMULATE_ID "GNU"
00031 # endif
00032   /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00033      except that a few beta releases use the old format with V=2021.  */
00034 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00035 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00036 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00037 #  if defined(__INTEL_COMPILER_UPDATE)
00038 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00039 #  else
00040 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00041 #  endif
00042 # else
00043 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00044 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00045    /* The third version component from --version is an update index,
00046       but no macro is provided for it.  */
00047 #  define COMPILER_VERSION_PATCH DEC(0)
00048 # endif
00049 # if defined(__INTEL_COMPILER_BUILD_DATE)
00050    /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00051 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00052 # endif
```

```
00053 # if defined(_MSC_VER)
00054    /* _MSC_VER = VVRR */
00055 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00056 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00057 # endif
00058 # if defined(__GNUC__)
00059 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00060 # elif defined(__GNUG__)
00061 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00062 # endif
00063 # if defined(__GNUC_MINOR__)
00064 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00065 # endif
00066 # if defined(__GNUC_PATCHLEVEL__)
00067 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00068 # endif
00069
00070 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00071 # define COMPILER_ID "IntelLLVM"
00072 #if defined(_MSC_VER)
00073 # define SIMULATE_ID "MSVC"
00074 #endif
00075 #if defined(__GNUC__)
00076 # define SIMULATE_ID "GNU"
00077 #endif
00078 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00079  * later.  Look for 6 digit vs. 8 digit version number to decide encoding.
00080  * VVVV is no smaller than the current year when a version is released.
00081  */
00082 #if __INTEL_LLVM_COMPILER < 1000000L
00083 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00084 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00085 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER    % 10)
00086 #else
00087 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00088 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00089 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER     % 100)
00090 #endif
00091 #if defined(_MSC_VER)
00092   /* _MSC_VER = VVRR */
00093 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00094 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00095 #endif
00096 #if defined(__GNUC__)
00097 # define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00098 #elif defined(__GNUG__)
00099 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00100 #endif
00101 #if defined(__GNUC_MINOR__)
00102 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00103 #endif
00104 #if defined(__GNUC_PATCHLEVEL__)
00105 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00106 #endif
00107
00108 #elif defined(__PATHCC__)
00109 # define COMPILER_ID "PathScale"
00110 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00111 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00112 # if defined(__PATHCC_PATCHLEVEL__)
00113 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00114 # endif
00115
00116 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00117 # define COMPILER_ID "Embarcadero"
00118 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00119 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00120 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00121
00122 #elif defined(__BORLANDC__)
00123 # define COMPILER_ID "Borland"
00124   /* __BORLANDC__ = 0xVRR */
00125 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00126 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00127
00128 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00129 # define COMPILER_ID "Watcom"
00130    /* __WATCOMC__ = VVRR */
00131 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00132 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00133 # if (__WATCOMC__ % 10) > 0
00134 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00135 # endif
00136
00137 #elif defined(__WATCOMC__)
00138 # define COMPILER_ID "OpenWatcom"
00139    /* __WATCOMC__ = VVRP + 1100 */
```

```
00140 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00141 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00142 # if (__WATCOMC__ % 10) > 0
00143 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00144 # endif
00145
00146 #elif defined(__SUNPRO_C)
00147 # define COMPILER_ID "SunPro"
00148 # if __SUNPRO_C >= 0x5100
00149    /* __SUNPRO_C = 0xVRRP */
00150 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C>>12)
00151 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C>>4 & 0xFF)
00152 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00153 # else
00154    /* __SUNPRO_CC = 0xVRP */
00155 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C>>8)
00156 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C>>4 & 0xF)
00157 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00158 # endif
00159
00160 #elif defined(__HP_cc)
00161 # define COMPILER_ID "HP"
00162   /* __HP_cc = VVRRPP */
00163 # define COMPILER_VERSION_MAJOR DEC(__HP_cc/10000)
00164 # define COMPILER_VERSION_MINOR DEC(__HP_cc/100 % 100)
00165 # define COMPILER_VERSION_PATCH DEC(__HP_cc     % 100)
00166
00167 #elif defined(__DECC)
00168 # define COMPILER_ID "Compaq"
00169   /* __DECC_VER = VVRRTPPPP */
00170 # define COMPILER_VERSION_MAJOR DEC(__DECC_VER/10000000)
00171 # define COMPILER_VERSION_MINOR DEC(__DECC_VER/100000  % 100)
00172 # define COMPILER_VERSION_PATCH DEC(__DECC_VER         % 10000)
00173
00174 #elif defined(__IBMC__) && defined(__COMPILER_VER__)
00175 # define COMPILER_ID "zOS"
00176   /* __IBMC__ = VRP */
00177 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00178 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00180
00181 #elif defined(__open_xl__) && defined(__clang__)
00182 # define COMPILER_ID "IBMClang"
00183 # define COMPILER_VERSION_MAJOR DEC(__open_xl_version__)
00184 # define COMPILER_VERSION_MINOR DEC(__open_xl_release__)
00185 # define COMPILER_VERSION_PATCH DEC(__open_xl_modification__)
00186 # define COMPILER_VERSION_TWEAK DEC(__open_xl_ptf_fix_level__)
00187
00188
00189 #elif defined(__ibmxl__) && defined(__clang__)
00190 # define COMPILER_ID "XLClang"
00191 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00192 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00193 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00194 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00195
00196
00197 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ >= 800
00198 # define COMPILER_ID "XL"
00199   /* __IBMC__ = VRP */
00200 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00201 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00202 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00203
00204 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ < 800
00205 # define COMPILER_ID "VisualAge"
00206   /* __IBMC__ = VRP */
00207 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00208 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00209 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00210
00211 #elif defined(__NVCOMPILER)
00212 # define COMPILER_ID "NVHPC"
00213 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00214 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00215 # if defined(__NVCOMPILER_PATCHLEVEL__)
00216 #  define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00217 # endif
00218
00219 #elif defined(__PGI)
00220 # define COMPILER_ID "PGI"
00221 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00222 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00223 # if defined(__PGIC_PATCHLEVEL__)
00224 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00225 # endif
00226
```

```
00227 #elif defined(__clang__) && defined(__cray__)
00228 # define COMPILER_ID "CrayClang"
00229 # define COMPILER_VERSION_MAJOR DEC(__cray_major__)
00230 # define COMPILER_VERSION_MINOR DEC(__cray_minor__)
00231 # define COMPILER_VERSION_PATCH DEC(__cray_patchlevel__)
00232 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00233
00234
00235 #elif defined(_CRAYC)
00236 # define COMPILER_ID "Cray"
00237 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00238 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00239
00240 #elif defined(__TI_COMPILER_VERSION__)
00241 # define COMPILER_ID "TI"
00242   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00243 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00244 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00245 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00246
00247 #elif defined(__CLANG_FUJITSU)
00248 # define COMPILER_ID "FujitsuClang"
00249 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00250 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00251 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00252 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00253
00254
00255 #elif defined(__FUJITSU)
00256 # define COMPILER_ID "Fujitsu"
00257 # if defined(__FCC_version__)
00258 #   define COMPILER_VERSION __FCC_version__
00259 # elif defined(__FCC_major__)
00260 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00261 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00262 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00263 # endif
00264 # if defined(__fcc_version)
00265 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00266 # elif defined(__FCC_VERSION)
00267 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00268 # endif
00269
00270
00271 #elif defined(__ghs__)
00272 # define COMPILER_ID "GHS"
00273 /* __GHS_VERSION_NUMBER = VVVVRP */
00274 # ifdef __GHS_VERSION_NUMBER
00275 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00276 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00277 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00278 # endif
00279
00280 #elif defined(__TASKING__)
00281 # define COMPILER_ID "Tasking"
00282   # define COMPILER_VERSION_MAJOR DEC(__VERSION__/1000)
00283   # define COMPILER_VERSION_MINOR DEC(__VERSION__ % 100)
00284 # define COMPILER_VERSION_INTERNAL DEC(__VERSION__)
00285
00286 #elif defined(__ORANGEC__)
00287 # define COMPILER_ID "OrangeC"
00288 # define COMPILER_VERSION_MAJOR DEC(__ORANGEC_MAJOR__)
00289 # define COMPILER_VERSION_MINOR DEC(__ORANGEC_MINOR__)
00290 # define COMPILER_VERSION_PATCH DEC(__ORANGEC_PATCHLEVEL__)
00291
00292 #elif defined(__TINYC__)
00293 # define COMPILER_ID "TinyCC"
00294
00295 #elif defined(__BCC__)
00296 # define COMPILER_ID "Bruce"
00297
00298 #elif defined(__SCO_VERSION__)
00299 # define COMPILER_ID "SCO"
00300
00301 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00302 # define COMPILER_ID "ARMCC"
00303 #if __ARMCC_VERSION >= 1000000
00304   /* __ARMCC_VERSION = VRRPPPP */
00305   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00306   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00307   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00308 #else
00309   /* __ARMCC_VERSION = VRPPPP */
00310   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00311   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00312   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00313 #endif
```

```
00314
00315
00316 #elif defined(__clang__) && defined(__apple_build_version__)
00317 # define COMPILER_ID "AppleClang"
00318 # if defined(_MSC_VER)
00319 #  define SIMULATE_ID "MSVC"
00320 # endif
00321 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00322 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00323 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00324 # if defined(_MSC_VER)
00325    /* _MSC_VER = VVRR */
00326 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00327 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00328 # endif
00329 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00330
00331 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00332 # define COMPILER_ID "ARMClang"
00333   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00334   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00335   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION/100   % 100)
00336 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00337
00338 #elif defined(__clang__) && defined(__ti__)
00339 # define COMPILER_ID "TIClang"
00340   # define COMPILER_VERSION_MAJOR DEC(__ti_major__)
00341   # define COMPILER_VERSION_MINOR DEC(__ti_minor__)
00342   # define COMPILER_VERSION_PATCH DEC(__ti_patchlevel__)
00343 # define COMPILER_VERSION_INTERNAL DEC(__ti_version__)
00344
00345 #elif defined(__clang__)
00346 # define COMPILER_ID "Clang"
00347 # if defined(_MSC_VER)
00348 #  define SIMULATE_ID "MSVC"
00349 # endif
00350 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00351 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00352 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00353 # if defined(_MSC_VER)
00354    /* _MSC_VER = VVRR */
00355 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00356 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00357 # endif
00358
00359 #elif defined(__LCC__) && (defined(__GNUC__) || defined(__GNUG__) || defined(__MCST__))
00360 # define COMPILER_ID "LCC"
00361 # define COMPILER_VERSION_MAJOR DEC(__LCC__ / 100)
00362 # define COMPILER_VERSION_MINOR DEC(__LCC__ % 100)
00363 # if defined(__LCC_MINOR__)
00364 #  define COMPILER_VERSION_PATCH DEC(__LCC_MINOR__)
00365 # endif
00366 # if defined(__GNUC__) && defined(__GNUC_MINOR__)
00367 #  define SIMULATE_ID "GNU"
00368 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00369 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00370 #  if defined(__GNUC_PATCHLEVEL__)
00371 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00372 #  endif
00373 # endif
00374
00375 #elif defined(__GNUC__)
00376 # define COMPILER_ID "GNU"
00377 # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00378 # if defined(__GNUC_MINOR__)
00379 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00380 # endif
00381 # if defined(__GNUC_PATCHLEVEL__)
00382 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00383 # endif
00384
00385 #elif defined(_MSC_VER)
00386 # define COMPILER_ID "MSVC"
00387   /* _MSC_VER = VVRR */
00388 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00389 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00390 # if defined(_MSC_FULL_VER)
00391 #  if _MSC_VER >= 1400
00392     /* _MSC_FULL_VER = VVRRPPPPP */
00393 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00394 #  else
00395     /* _MSC_FULL_VER = VVRRPPPP */
00396 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00397 #  endif
00398 # endif
00399 # if defined(_MSC_BUILD)
00400 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
```

```
00401 # endif
00402
00403 #elif defined(_ADI_COMPILER)
00404 # define COMPILER_ID "ADSP"
00405 #if defined(__VERSIONNUM__)
00406   /* __VERSIONNUM__ = 0xVVRRPPTT */
00407 #  define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ » 24 & 0xFF)
00408 #  define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ » 16 & 0xFF)
00409 #  define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ » 8 & 0xFF)
00410 #  define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00411 #endif
00412
00413 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00414 # define COMPILER_ID "IAR"
00415 # if defined(__VER__) && defined(__ICCARM__)
00416 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00417 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00418 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00419 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00420 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
      defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
      defined(__ICC8051__) || defined(__ICCSTM8__))
00421 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00422 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00423 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00424 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00425 # endif
00426
00427 #elif defined(__SDCC_VERSION_MAJOR) || defined(SDCC)
00428 # define COMPILER_ID "SDCC"
00429 # if defined(__SDCC_VERSION_MAJOR)
00430 #  define COMPILER_VERSION_MAJOR DEC(__SDCC_VERSION_MAJOR)
00431 #  define COMPILER_VERSION_MINOR DEC(__SDCC_VERSION_MINOR)
00432 #  define COMPILER_VERSION_PATCH DEC(__SDCC_VERSION_PATCH)
00433 # else
00434   /* SDCC = VRP */
00435 #  define COMPILER_VERSION_MAJOR DEC(SDCC/100)
00436 #  define COMPILER_VERSION_MINOR DEC(SDCC/10 % 10)
00437 #  define COMPILER_VERSION_PATCH DEC(SDCC    % 10)
00438 # endif
00439
00440
00441 /* These compilers are either not known or too old to define an
00442   identification macro.  Try to identify the platform and guess that
00443   it is the native compiler.  */
00444 #elif defined(__hpux) || defined(__hpua)
00445 # define COMPILER_ID "HP"
00446
00447 #else /* unknown compiler */
00448 # define COMPILER_ID ""
00449 #endif
00450
00451 /* Construct the string literal in pieces to prevent the source from
00452    getting matched.  Store it in a pointer rather than an array
00453    because some compilers will just produce instructions to fill the
00454    array rather than assigning a pointer to a static array.  */
00455 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00456 #ifdef SIMULATE_ID
00457 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00458 #endif
00459
00460 #ifdef __QNXNTO__
00461 char const* qnxnto = "INFO" ":" "qnxnto[]";
00462 #endif
00463
00464 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00465 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00466 #endif
00467
00468 #define STRINGIFY_HELPER(X) #X
00469 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00470
00471 /* Identify known platforms by name.  */
00472 #if defined(__linux) || defined(__linux__) || defined(linux)
00473 # define PLATFORM_ID "Linux"
00474
00475 #elif defined(__MSYS__)
00476 # define PLATFORM_ID "MSYS"
00477
00478 #elif defined(__CYGWIN__)
00479 # define PLATFORM_ID "Cygwin"
00480
00481 #elif defined(__MINGW32__)
00482 # define PLATFORM_ID "MinGW"
00483
00484 #elif defined(__APPLE__)
00485 # define PLATFORM_ID "Darwin"
```

```
00486
00487 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00488 # define PLATFORM_ID "Windows"
00489
00490 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00491 # define PLATFORM_ID "FreeBSD"
00492
00493 #elif defined(__NetBSD__) || defined(__NetBSD)
00494 # define PLATFORM_ID "NetBSD"
00495
00496 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00497 # define PLATFORM_ID "OpenBSD"
00498
00499 #elif defined(__sun) || defined(sun)
00500 # define PLATFORM_ID "SunOS"
00501
00502 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00503 # define PLATFORM_ID "AIX"
00504
00505 #elif defined(__hpux) || defined(__hpux__)
00506 # define PLATFORM_ID "HP-UX"
00507
00508 #elif defined(__HAIKU__)
00509 # define PLATFORM_ID "Haiku"
00510
00511 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00512 # define PLATFORM_ID "BeOS"
00513
00514 #elif defined(__QNX__) || defined(__QNXNTO__)
00515 # define PLATFORM_ID "QNX"
00516
00517 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00518 # define PLATFORM_ID "Tru64"
00519
00520 #elif defined(__riscos) || defined(__riscos__)
00521 # define PLATFORM_ID "RISCos"
00522
00523 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00524 # define PLATFORM_ID "SINIX"
00525
00526 #elif defined(__UNIX_SV__)
00527 # define PLATFORM_ID "UNIX_SV"
00528
00529 #elif defined(__bsdos__)
00530 # define PLATFORM_ID "BSDOS"
00531
00532 #elif defined(_MPRAS) || defined(MPRAS)
00533 # define PLATFORM_ID "MP-RAS"
00534
00535 #elif defined(__osf) || defined(__osf__)
00536 # define PLATFORM_ID "OSF1"
00537
00538 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00539 # define PLATFORM_ID "SCO_SV"
00540
00541 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00542 # define PLATFORM_ID "ULTRIX"
00543
00544 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00545 # define PLATFORM_ID "Xenix"
00546
00547 #elif defined(__WATCOMC__)
00548 # if defined(__LINUX__)
00549 #  define PLATFORM_ID "Linux"
00550
00551 # elif defined(__DOS__)
00552 #  define PLATFORM_ID "DOS"
00553
00554 # elif defined(__OS2__)
00555 #  define PLATFORM_ID "OS2"
00556
00557 # elif defined(__WINDOWS__)
00558 #  define PLATFORM_ID "Windows3x"
00559
00560 # elif defined(__VXWORKS__)
00561 #  define PLATFORM_ID "VxWorks"
00562
00563 # else /* unknown platform */
00564 #  define PLATFORM_ID
00565 # endif
00566
00567 #elif defined(__INTEGRITY)
00568 # if defined(INT_178B)
00569 #  define PLATFORM_ID "Integrity178"
00570
00571 # else /* regular Integrity */
00572 #  define PLATFORM_ID "Integrity"
```

```
00573 # endif
00574
00575 # elif defined(_ADI_COMPILER)
00576 #  define PLATFORM_ID "ADSP"
00577
00578 #else /* unknown platform */
00579 # define PLATFORM_ID
00580
00581 #endif
00582
00583 /* For windows compilers MSVC and Intel we can determine
00584    the architecture of the compiler being used.  This is because
00585    the compilers do not have flags that can change the architecture,
00586    but rather depend on which compiler is being used
00587 */
00588 #if defined(_WIN32) && defined(_MSC_VER)
00589 # if defined(_M_IA64)
00590 #  define ARCHITECTURE_ID "IA64"
00591
00592 # elif defined(_M_ARM64EC)
00593 #  define ARCHITECTURE_ID "ARM64EC"
00594
00595 # elif defined(_M_X64) || defined(_M_AMD64)
00596 #  define ARCHITECTURE_ID "x64"
00597
00598 # elif defined(_M_IX86)
00599 #  define ARCHITECTURE_ID "X86"
00600
00601 # elif defined(_M_ARM64)
00602 #  define ARCHITECTURE_ID "ARM64"
00603
00604 # elif defined(_M_ARM)
00605 #  if _M_ARM == 4
00606 #   define ARCHITECTURE_ID "ARMV4I"
00607 #  elif _M_ARM == 5
00608 #   define ARCHITECTURE_ID "ARMV5I"
00609 #  else
00610 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00611 #  endif
00612
00613 # elif defined(_M_MIPS)
00614 #  define ARCHITECTURE_ID "MIPS"
00615
00616 # elif defined(_M_SH)
00617 #  define ARCHITECTURE_ID "SHx"
00618
00619 # else /* unknown architecture */
00620 #  define ARCHITECTURE_ID ""
00621 # endif
00622
00623 #elif defined(__WATCOMC__)
00624 # if defined(_M_I86)
00625 #  define ARCHITECTURE_ID "I86"
00626
00627 # elif defined(_M_IX86)
00628 #  define ARCHITECTURE_ID "X86"
00629
00630 # else /* unknown architecture */
00631 #  define ARCHITECTURE_ID ""
00632 # endif
00633
00634 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00635 # if defined(__ICCARM__)
00636 #  define ARCHITECTURE_ID "ARM"
00637
00638 # elif defined(__ICCRX__)
00639 #  define ARCHITECTURE_ID "RX"
00640
00641 # elif defined(__ICCRH850__)
00642 #  define ARCHITECTURE_ID "RH850"
00643
00644 # elif defined(__ICCRL78__)
00645 #  define ARCHITECTURE_ID "RL78"
00646
00647 # elif defined(__ICCRISCV__)
00648 #  define ARCHITECTURE_ID "RISCV"
00649
00650 # elif defined(__ICCAVR__)
00651 #  define ARCHITECTURE_ID "AVR"
00652
00653 # elif defined(__ICC430__)
00654 #  define ARCHITECTURE_ID "MSP430"
00655
00656 # elif defined(__ICCV850__)
00657 #  define ARCHITECTURE_ID "V850"
00658
00659 # elif defined(__ICC8051__)
```

```
00660 #  define ARCHITECTURE_ID "8051"
00661
00662 # elif defined(__ICCSTM8__)
00663 #  define ARCHITECTURE_ID "STM8"
00664
00665 # else /* unknown architecture */
00666 #  define ARCHITECTURE_ID ""
00667 # endif
00668
00669 #elif defined(__ghs__)
00670 # if defined(__PPC64__)
00671 #  define ARCHITECTURE_ID "PPC64"
00672
00673 # elif defined(__ppc__)
00674 #  define ARCHITECTURE_ID "PPC"
00675
00676 # elif defined(__ARM__)
00677 #  define ARCHITECTURE_ID "ARM"
00678
00679 # elif defined(__x86_64__)
00680 #  define ARCHITECTURE_ID "x64"
00681
00682 # elif defined(__i386__)
00683 #  define ARCHITECTURE_ID "X86"
00684
00685 # else /* unknown architecture */
00686 #  define ARCHITECTURE_ID ""
00687 # endif
00688
00689 #elif defined(__clang__) && defined(__ti__)
00690 # if defined(__ARM_ARCH)
00691 #  define ARCHITECTURE_ID "Arm"
00692
00693 # else /* unknown architecture */
00694 #  define ARCHITECTURE_ID ""
00695 # endif
00696
00697 #elif defined(__TI_COMPILER_VERSION__)
00698 # if defined(__TI_ARM__)
00699 #  define ARCHITECTURE_ID "ARM"
00700
00701 # elif defined(__MSP430__)
00702 #  define ARCHITECTURE_ID "MSP430"
00703
00704 # elif defined(__TMS320C28XX__)
00705 #  define ARCHITECTURE_ID "TMS320C28x"
00706
00707 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00708 #  define ARCHITECTURE_ID "TMS320C6x"
00709
00710 # else /* unknown architecture */
00711 #  define ARCHITECTURE_ID ""
00712 # endif
00713
00714 # elif defined(__ADSPSHARC__)
00715 #  define ARCHITECTURE_ID "SHARC"
00716
00717 # elif defined(__ADSPBLACKFIN__)
00718 #  define ARCHITECTURE_ID "Blackfin"
00719
00720 #elif defined(__TASKING__)
00721
00722 # if defined(__CTC__) || defined(__CPTC__)
00723 #  define ARCHITECTURE_ID "TriCore"
00724
00725 # elif defined(__CMCS__)
00726 #  define ARCHITECTURE_ID "MCS"
00727
00728 # elif defined(__CARM__)
00729 #  define ARCHITECTURE_ID "ARM"
00730
00731 # elif defined(__CARC__)
00732 #  define ARCHITECTURE_ID "ARC"
00733
00734 # elif defined(__C51__)
00735 #  define ARCHITECTURE_ID "8051"
00736
00737 # elif defined(__CPCP__)
00738 #  define ARCHITECTURE_ID "PCP"
00739
00740 # else
00741 #  define ARCHITECTURE_ID ""
00742 # endif
00743
00744 #else
00745 #  define ARCHITECTURE_ID
00746 #endif
```

```
00747
00748 /* Convert integer to decimal digit literals.  */
00749 #define DEC(n)                       \
00750   ('0' + (((n) / 10000000)%10)), \
00751   ('0' + (((n) / 1000000)%10)),  \
00752   ('0' + (((n) / 100000)%10)),   \
00753   ('0' + (((n) / 10000)%10)),    \
00754   ('0' + (((n) / 1000)%10)),     \
00755   ('0' + (((n) / 100)%10)),      \
00756   ('0' + (((n) / 10)%10)),       \
00757   ('0' +  ((n) % 10))
00758
00759 /* Convert integer to hex digit literals.  */
00760 #define HEX(n)             \
00761   ('0' + ((n)>>28 & 0xF)), \
00762   ('0' + ((n)>>24 & 0xF)), \
00763   ('0' + ((n)>>20 & 0xF)), \
00764   ('0' + ((n)>>16 & 0xF)), \
00765   ('0' + ((n)>>12 & 0xF)), \
00766   ('0' + ((n)>>8  & 0xF)), \
00767   ('0' + ((n)>>4  & 0xF)), \
00768   ('0' + ((n)     & 0xF))
00769
00770 /* Construct a string literal encoding the version number. */
00771 #ifdef COMPILER_VERSION
00772 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "]";
00773
00774 /* Construct a string literal encoding the version number components. */
00775 #elif defined(COMPILER_VERSION_MAJOR)
00776 char const info_version[] = {
00777   'I', 'N', 'F', 'O', ':',
00778   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00779   COMPILER_VERSION_MAJOR,
00780 # ifdef COMPILER_VERSION_MINOR
00781   '.', COMPILER_VERSION_MINOR,
00782 #  ifdef COMPILER_VERSION_PATCH
00783    '.', COMPILER_VERSION_PATCH,
00784 #   ifdef COMPILER_VERSION_TWEAK
00785     '.', COMPILER_VERSION_TWEAK,
00786 #   endif
00787 #  endif
00788 # endif
00789   ']','\0'};
00790 #endif
00791
00792 /* Construct a string literal encoding the internal version number. */
00793 #ifdef COMPILER_VERSION_INTERNAL
00794 char const info_version_internal[] = {
00795   'I', 'N', 'F', 'O', ':',
00796   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00797   'i','n','t','e','r','n','a','l','[',
00798   COMPILER_VERSION_INTERNAL,']','\0'};
00799 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00800 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
    COMPILER_VERSION_INTERNAL_STR "]";
00801 #endif
00802
00803 /* Construct a string literal encoding the version number components. */
00804 #ifdef SIMULATE_VERSION_MAJOR
00805 char const info_simulate_version[] = {
00806   'I', 'N', 'F', 'O', ':',
00807   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00808   SIMULATE_VERSION_MAJOR,
00809 # ifdef SIMULATE_VERSION_MINOR
00810   '.', SIMULATE_VERSION_MINOR,
00811 #  ifdef SIMULATE_VERSION_PATCH
00812    '.', SIMULATE_VERSION_PATCH,
00813 #   ifdef SIMULATE_VERSION_TWEAK
00814     '.', SIMULATE_VERSION_TWEAK,
00815 #   endif
00816 #  endif
00817 # endif
00818   ']','\0'};
00819 #endif
00820
00821 /* Construct the string literal in pieces to prevent the source from
00822    getting matched.  Store it in a pointer rather than an array
00823    because some compilers will just produce instructions to fill the
00824    array rather than assigning a pointer to a static array.  */
00825 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00826 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00827
00828
00829
00830 #define C_STD_99 199901L
00831 #define C_STD_11 201112L
00832 #define C_STD_17 201710L
```

```
00833 #define C_STD_23 202311L
00834
00835 #ifdef __STDC_VERSION__
00836 #  define C_STD __STDC_VERSION__
00837 #endif
00838
00839 #if !defined(__STDC__) && !defined(__clang__)
00840 # if defined(_MSC_VER) || defined(__ibmxl__) || defined(__IBMC__)
00841 #  define C_VERSION "90"
00842 # else
00843 #  define C_VERSION
00844 # endif
00845 #elif C_STD > C_STD_17
00846 # define C_VERSION "23"
00847 #elif C_STD > C_STD_11
00848 # define C_VERSION "17"
00849 #elif C_STD > C_STD_99
00850 # define C_VERSION "11"
00851 #elif C_STD >= C_STD_99
00852 # define C_VERSION "99"
00853 #else
00854 # define C_VERSION "90"
00855 #endif
00856 const char* info_language_standard_default =
00857   "INFO" ":" "standard_default[" C_VERSION "]";
00858
00859 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00860 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlC__) ||           \
00861      defined(__TI_COMPILER_VERSION__)) &&                                     \
00862   !defined(__STRICT_ANSI__)
00863   "ON"
00864 #else
00865   "OFF"
00866 #endif
00867 "]";
00868
00869 /*--------------------------------------------------------------------------*/
00870
00871 #ifdef ID_VOID_MAIN
00872 void main() {}
00873 #else
00874 # if defined(__CLASSIC_C__)
00875 int main(argc, argv) int argc; char *argv[];
00876 # else
00877 int main(int argc, char* argv[])
00878 # endif
00879 {
00880   int require = 0;
00881   require += info_compiler[argc];
00882   require += info_platform[argc];
00883   require += info_arch[argc];
00884 #ifdef COMPILER_VERSION_MAJOR
00885   require += info_version[argc];
00886 #endif
00887 #ifdef COMPILER_VERSION_INTERNAL
00888   require += info_version_internal[argc];
00889 #endif
00890 #ifdef SIMULATE_ID
00891   require += info_simulate[argc];
00892 #endif
00893 #ifdef SIMULATE_VERSION_MAJOR
00894   require += info_simulate_version[argc];
00895 #endif
00896 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00897   require += info_cray[argc];
00898 #endif
00899   require += info_language_standard_default[argc];
00900   require += info_language_extensions_default[argc];
00901   (void)argv;
00902   return require;
00903 }
00904 #endif
```

6.3 build/CMakeFiles/3.31.0/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x)

- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X)
- #define STRINGIFY(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_STD_99 199901L
- #define C_STD_11 201112L
- #define C_STD_17 201710L
- #define C_STD_23 202311L
- #define C_VERSION

Functions

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

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

6.3.1 Macro Definition Documentation

6.3.1.1 __has_include

```
#define __has_include(
              x)
```

Value:

```
0
```

Definition at line 17 of file CMakeCCompilerId.c.

6.3.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 745 of file CMakeCCompilerId.c.

6.3.1.3 C_STD_11

```
#define C_STD_11 201112L
```

Definition at line 831 of file CMakeCCompilerId.c.

6.3.1.4 C_STD_17

```
#define C_STD_17 201710L
```

Definition at line 832 of file CMakeCCompilerId.c.

6.3.1.5 C_STD_23

```
#define C_STD_23 202311L
```

Definition at line 833 of file CMakeCCompilerId.c.

6.3.1.6 C_STD_99

```
#define C_STD_99 199901L
```

Definition at line 830 of file CMakeCCompilerId.c.

6.3.1.7 C_VERSION

```
#define C_VERSION
```

Definition at line 843 of file CMakeCCompilerId.c.

6.3.1.8 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 448 of file CMakeCCompilerId.c.

6.3.1.9 DEC

```
#define DEC(
              n)
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 749 of file CMakeCCompilerId.c.

6.3.1.10 HEX

```
#define HEX(
              n)
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 760 of file CMakeCCompilerId.c.

6.3.1.11 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 579 of file CMakeCCompilerId.c.

6.3.1.12 STRINGIFY

```
#define STRINGIFY(
              X)
```

Value:

```
STRINGIFY_HELPER(X)
```

Definition at line 469 of file CMakeCCompilerId.c.

6.3.1.13 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X)
```

Value:

```
#X
```

Definition at line 468 of file CMakeCCompilerId.c.

6.3.2 Function Documentation

6.3.2.1 main()

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

Definition at line 877 of file CMakeCCompilerId.c.

6.3.3 Variable Documentation

6.3.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 826 of file CMakeCCompilerId.c.

6.3.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 455 of file CMakeCCompilerId.c.

6.3.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["



  "OFF"

"]"
```

Definition at line 859 of file CMakeCCompilerId.c.

6.3.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 856 of file CMakeCCompilerId.c.

6.3.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 825 of file CMakeCCompilerId.c.

6.4 CMakeCCompilerId.c

Go to the documentation of this file.

```
00001 #ifdef __cplusplus
00002 # error "A C++ compiler has been selected for C."
00003 #endif
00004
00005 #if defined(__18CXX)
00006 # define ID_VOID_MAIN
00007 #endif
00008 #if defined(__CLASSIC_C__)
00009 /* cv-qualifiers did not exist in K&R C */
00010 # define const
00011 # define volatile
00012 #endif
00013
00014 #if !defined(__has_include)
00015 /* If the compiler does not have __has_include, pretend the answer is
00016    always no.  */
00017 #  define __has_include(x) 0
00018 #endif
00019
00020
00021 /* Version number components: V=Version, R=Revision, P=Patch
00022    Version date components:   YYYY=Year, MM=Month,   DD=Day  */
00023
00024 #if defined(__INTEL_COMPILER) || defined(__ICC)
00025 # define COMPILER_ID "Intel"
00026 # if defined(_MSC_VER)
00027 #  define SIMULATE_ID "MSVC"
00028 # endif
00029 # if defined(__GNUC__)
00030 #  define SIMULATE_ID "GNU"
00031 # endif
00032   /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00033      except that a few beta releases use the old format with V=2021.  */
00034 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00035 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00036 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00037 #  if defined(__INTEL_COMPILER_UPDATE)
00038 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00039 #  else
00040 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00041 #  endif
00042 # else
00043 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00044 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00045    /* The third version component from --version is an update index,
00046       but no macro is provided for it.  */
00047 #  define COMPILER_VERSION_PATCH DEC(0)
00048 # endif
00049 # if defined(__INTEL_COMPILER_BUILD_DATE)
00050    /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00051 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00052 # endif
00053 # if defined(_MSC_VER)
00054    /* _MSC_VER = VVRR */
00055 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00056 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00057 # endif
00058 # if defined(__GNUC__)
00059 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00060 # elif defined(__GNUG__)
00061 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00062 # endif
00063 # if defined(__GNUC_MINOR__)
00064 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00065 # endif
00066 # if defined(__GNUC_PATCHLEVEL__)
00067 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00068 # endif
00069
00070 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00071 # define COMPILER_ID "IntelLLVM"
00072 #if defined(_MSC_VER)
00073 # define SIMULATE_ID "MSVC"
00074 #endif
00075 #if defined(__GNUC__)
00076 # define SIMULATE_ID "GNU"
00077 #endif
00078 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00079  * later.  Look for 6 digit vs. 8 digit version number to decide encoding.
00080  * VVVV is no smaller than the current year when a version is released.
00081  */
00082 #if __INTEL_LLVM_COMPILER < 1000000L
```

```
00083 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00084 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00085 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER    % 10)
00086 #else
00087 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00088 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00089 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER     % 100)
00090 #endif
00091 #if defined(_MSC_VER)
00092   /* _MSC_VER = VVRR */
00093 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00094 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00095 #endif
00096 #if defined(__GNUC__)
00097 # define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00098 #elif defined(__GNUG__)
00099 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00100 #endif
00101 #if defined(__GNUC_MINOR__)
00102 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00103 #endif
00104 #if defined(__GNUC_PATCHLEVEL__)
00105 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00106 #endif
00107
00108 #elif defined(__PATHCC__)
00109 # define COMPILER_ID "PathScale"
00110 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00111 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00112 # if defined(__PATHCC_PATCHLEVEL__)
00113 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00114 # endif
00115
00116 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00117 # define COMPILER_ID "Embarcadero"
00118 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00119 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00120 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00121
00122 #elif defined(__BORLANDC__)
00123 # define COMPILER_ID "Borland"
00124   /* __BORLANDC__ = 0xVRR */
00125 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00126 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00127
00128 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00129 # define COMPILER_ID "Watcom"
00130    /* __WATCOMC__ = VVRR */
00131 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00132 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00133 # if (__WATCOMC__ % 10) > 0
00134 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00135 # endif
00136
00137 #elif defined(__WATCOMC__)
00138 # define COMPILER_ID "OpenWatcom"
00139    /* __WATCOMC__ = VVRP + 1100 */
00140 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00141 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00142 # if (__WATCOMC__ % 10) > 0
00143 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00144 # endif
00145
00146 #elif defined(__SUNPRO_C)
00147 # define COMPILER_ID "SunPro"
00148 # if __SUNPRO_C >= 0x5100
00149    /* __SUNPRO_C = 0xVRRP */
00150 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C»12)
00151 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C»4 & 0xFF)
00152 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00153 # else
00154    /* __SUNPRO_CC = 0xVRP */
00155 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C»8)
00156 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C»4 & 0xF)
00157 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00158 # endif
00159
00160 #elif defined(__HP_cc)
00161 # define COMPILER_ID "HP"
00162   /* __HP_cc = VVRRPP */
00163 # define COMPILER_VERSION_MAJOR DEC(__HP_cc/10000)
00164 # define COMPILER_VERSION_MINOR DEC(__HP_cc/100 % 100)
00165 # define COMPILER_VERSION_PATCH DEC(__HP_cc     % 100)
00166
00167 #elif defined(__DECC)
00168 # define COMPILER_ID "Compaq"
00169   /* __DECC_VER = VVRRTPPPP */
```

```
00170 # define COMPILER_VERSION_MAJOR DEC(__DECC_VER/10000000)
00171 # define COMPILER_VERSION_MINOR DEC(__DECC_VER/100000  % 100)
00172 # define COMPILER_VERSION_PATCH DEC(__DECC_VER         % 10000)
00173
00174 #elif defined(__IBMC__) && defined(__COMPILER_VER__)
00175 # define COMPILER_ID "zOS"
00176   /* __IBMC__ = VRP */
00177 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00178 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00180
00181 #elif defined(__open_xl__) && defined(__clang__)
00182 # define COMPILER_ID "IBMClang"
00183 # define COMPILER_VERSION_MAJOR DEC(__open_xl_version__)
00184 # define COMPILER_VERSION_MINOR DEC(__open_xl_release__)
00185 # define COMPILER_VERSION_PATCH DEC(__open_xl_modification__)
00186 # define COMPILER_VERSION_TWEAK DEC(__open_xl_ptf_fix_level__)
00187
00188
00189 #elif defined(__ibmxl__) && defined(__clang__)
00190 # define COMPILER_ID "XLClang"
00191 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00192 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00193 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00194 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00195
00196
00197 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ >= 800
00198 # define COMPILER_ID "XL"
00199   /* __IBMC__ = VRP */
00200 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00201 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00202 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00203
00204 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ < 800
00205 # define COMPILER_ID "VisualAge"
00206   /* __IBMC__ = VRP */
00207 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00208 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00209 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00210
00211 #elif defined(__NVCOMPILER)
00212 # define COMPILER_ID "NVHPC"
00213 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00214 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00215 # if defined(__NVCOMPILER_PATCHLEVEL__)
00216 #  define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00217 # endif
00218
00219 #elif defined(__PGI)
00220 # define COMPILER_ID "PGI"
00221 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00222 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00223 # if defined(__PGIC_PATCHLEVEL__)
00224 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00225 # endif
00226
00227 #elif defined(__clang__) && defined(__cray__)
00228 # define COMPILER_ID "CrayClang"
00229 # define COMPILER_VERSION_MAJOR DEC(__cray_major__)
00230 # define COMPILER_VERSION_MINOR DEC(__cray_minor__)
00231 # define COMPILER_VERSION_PATCH DEC(__cray_patchlevel__)
00232 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00233
00234
00235 #elif defined(_CRAYC)
00236 # define COMPILER_ID "Cray"
00237 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00238 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00239
00240 #elif defined(__TI_COMPILER_VERSION__)
00241 # define COMPILER_ID "TI"
00242   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00243 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00244 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00245 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00246
00247 #elif defined(__CLANG_FUJITSU)
00248 # define COMPILER_ID "FujitsuClang"
00249 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00250 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00251 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00252 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00253
00254
00255 #elif defined(__FUJITSU)
00256 # define COMPILER_ID "Fujitsu"
```

```
00257 # if defined(__FCC_version__)
00258 #   define COMPILER_VERSION __FCC_version__
00259 # elif defined(__FCC_major__)
00260 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00261 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00262 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00263 # endif
00264 # if defined(__fcc_version)
00265 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00266 # elif defined(__FCC_VERSION)
00267 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00268 # endif
00269
00270
00271 #elif defined(__ghs__)
00272 # define COMPILER_ID "GHS"
00273 /* __GHS_VERSION_NUMBER = VVVVRP */
00274 # ifdef __GHS_VERSION_NUMBER
00275 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00276 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00277 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00278 # endif
00279
00280 #elif defined(__TASKING__)
00281 # define COMPILER_ID "Tasking"
00282   # define COMPILER_VERSION_MAJOR DEC(__VERSION__/1000)
00283   # define COMPILER_VERSION_MINOR DEC(__VERSION__ % 100)
00284 # define COMPILER_VERSION_INTERNAL DEC(__VERSION__)
00285
00286 #elif defined(__ORANGEC__)
00287 # define COMPILER_ID "OrangeC"
00288 # define COMPILER_VERSION_MAJOR DEC(__ORANGEC_MAJOR__)
00289 # define COMPILER_VERSION_MINOR DEC(__ORANGEC_MINOR__)
00290 # define COMPILER_VERSION_PATCH DEC(__ORANGEC_PATCHLEVEL__)
00291
00292 #elif defined(__TINYC__)
00293 # define COMPILER_ID "TinyCC"
00294
00295 #elif defined(__BCC__)
00296 # define COMPILER_ID "Bruce"
00297
00298 #elif defined(__SCO_VERSION__)
00299 # define COMPILER_ID "SCO"
00300
00301 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00302 # define COMPILER_ID "ARMCC"
00303 #if __ARMCC_VERSION >= 1000000
00304   /* __ARMCC_VERSION = VRRPPPP */
00305   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00306   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00307   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00308 #else
00309   /* __ARMCC_VERSION = VRPPPP */
00310   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00311   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00312   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00313 #endif
00314
00315
00316 #elif defined(__clang__) && defined(__apple_build_version__)
00317 # define COMPILER_ID "AppleClang"
00318 # if defined(_MSC_VER)
00319 #  define SIMULATE_ID "MSVC"
00320 # endif
00321 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00322 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00323 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00324 # if defined(_MSC_VER)
00325   /* _MSC_VER = VVRR */
00326 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00327 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00328 # endif
00329 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00330
00331 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00332 # define COMPILER_ID "ARMClang"
00333   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00334   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00335   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION/100   % 100)
00336 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00337
00338 #elif defined(__clang__) && defined(__ti__)
00339 # define COMPILER_ID "TIClang"
00340   # define COMPILER_VERSION_MAJOR DEC(__ti_major__)
00341   # define COMPILER_VERSION_MINOR DEC(__ti_minor__)
00342   # define COMPILER_VERSION_PATCH DEC(__ti_patchlevel__)
00343 # define COMPILER_VERSION_INTERNAL DEC(__ti_version__)
```

```
00344
00345 #elif defined(__clang__)
00346 # define COMPILER_ID "Clang"
00347 # if defined(_MSC_VER)
00348 #  define SIMULATE_ID "MSVC"
00349 # endif
00350 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00351 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00352 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00353 # if defined(_MSC_VER)
00354    /* _MSC_VER = VVRR */
00355 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00356 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00357 # endif
00358
00359 #elif defined(__LCC__) && (defined(__GNUC__) || defined(__GNUG__) || defined(__MCST__))
00360 # define COMPILER_ID "LCC"
00361 # define COMPILER_VERSION_MAJOR DEC(__LCC__ / 100)
00362 # define COMPILER_VERSION_MINOR DEC(__LCC__ % 100)
00363 # if defined(__LCC_MINOR__)
00364 #  define COMPILER_VERSION_PATCH DEC(__LCC_MINOR__)
00365 # endif
00366 # if defined(__GNUC__) && defined(__GNUC_MINOR__)
00367 #  define SIMULATE_ID "GNU"
00368 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00369 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00370 #  if defined(__GNUC_PATCHLEVEL__)
00371 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00372 #  endif
00373 # endif
00374
00375 #elif defined(__GNUC__)
00376 # define COMPILER_ID "GNU"
00377 # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00378 # if defined(__GNUC_MINOR__)
00379 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00380 # endif
00381 # if defined(__GNUC_PATCHLEVEL__)
00382 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00383 # endif
00384
00385 #elif defined(_MSC_VER)
00386 # define COMPILER_ID "MSVC"
00387   /* _MSC_VER = VVRR */
00388 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00389 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00390 # if defined(_MSC_FULL_VER)
00391 #  if _MSC_VER >= 1400
00392     /* _MSC_FULL_VER = VVRRPPPPP */
00393 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00394 #  else
00395     /* _MSC_FULL_VER = VVRRPPPP */
00396 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00397 #  endif
00398 # endif
00399 # if defined(_MSC_BUILD)
00400 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00401 # endif
00402
00403 #elif defined(_ADI_COMPILER)
00404 # define COMPILER_ID "ADSP"
00405 #if defined(__VERSIONNUM__)
00406   /* __VERSIONNUM__ = 0xVVRRPTT */
00407 #  define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ » 24 & 0xFF)
00408 #  define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ » 16 & 0xFF)
00409 #  define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ » 8 & 0xFF)
00410 #  define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00411 #endif
00412
00413 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00414 # define COMPILER_ID "IAR"
00415 # if defined(__VER__) && defined(__ICCARM__)
00416 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00417 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00418 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00419 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00420 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
      defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
      defined(__ICC8051__) || defined(__ICCSTM8__))
00421 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00422 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00423 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00424 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00425 # endif
00426
00427 #elif defined(__SDCC_VERSION_MAJOR) || defined(SDCC)
00428 # define COMPILER_ID "SDCC"
```

```
00429 # if defined(__SDCC_VERSION_MAJOR)
00430 #  define COMPILER_VERSION_MAJOR DEC(__SDCC_VERSION_MAJOR)
00431 #  define COMPILER_VERSION_MINOR DEC(__SDCC_VERSION_MINOR)
00432 #  define COMPILER_VERSION_PATCH DEC(__SDCC_VERSION_PATCH)
00433 # else
00434   /* SDCC = VRP */
00435 #  define COMPILER_VERSION_MAJOR DEC(SDCC/100)
00436 #  define COMPILER_VERSION_MINOR DEC(SDCC/10 % 10)
00437 #  define COMPILER_VERSION_PATCH DEC(SDCC    % 10)
00438 # endif
00439
00440
00441 /* These compilers are either not known or too old to define an
00442   identification macro.  Try to identify the platform and guess that
00443   it is the native compiler.  */
00444 #elif defined(__hpux) || defined(__hpua)
00445 # define COMPILER_ID "HP"
00446
00447 #else /* unknown compiler */
00448 # define COMPILER_ID ""
00449 #endif
00450
00451 /* Construct the string literal in pieces to prevent the source from
00452    getting matched.  Store it in a pointer rather than an array
00453    because some compilers will just produce instructions to fill the
00454    array rather than assigning a pointer to a static array.  */
00455 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00456 #ifdef SIMULATE_ID
00457 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00458 #endif
00459
00460 #ifdef __QNXNTO__
00461 char const* qnxnto = "INFO" ":" "qnxnto[]";
00462 #endif
00463
00464 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00465 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00466 #endif
00467
00468 #define STRINGIFY_HELPER(X) #X
00469 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00470
00471 /* Identify known platforms by name.  */
00472 #if defined(__linux) || defined(__linux__) || defined(linux)
00473 # define PLATFORM_ID "Linux"
00474
00475 #elif defined(__MSYS__)
00476 # define PLATFORM_ID "MSYS"
00477
00478 #elif defined(__CYGWIN__)
00479 # define PLATFORM_ID "Cygwin"
00480
00481 #elif defined(__MINGW32__)
00482 # define PLATFORM_ID "MinGW"
00483
00484 #elif defined(__APPLE__)
00485 # define PLATFORM_ID "Darwin"
00486
00487 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00488 # define PLATFORM_ID "Windows"
00489
00490 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00491 # define PLATFORM_ID "FreeBSD"
00492
00493 #elif defined(__NetBSD__) || defined(__NetBSD)
00494 # define PLATFORM_ID "NetBSD"
00495
00496 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00497 # define PLATFORM_ID "OpenBSD"
00498
00499 #elif defined(__sun) || defined(sun)
00500 # define PLATFORM_ID "SunOS"
00501
00502 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00503 # define PLATFORM_ID "AIX"
00504
00505 #elif defined(__hpux) || defined(__hpux__)
00506 # define PLATFORM_ID "HP-UX"
00507
00508 #elif defined(__HAIKU__)
00509 # define PLATFORM_ID "Haiku"
00510
00511 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00512 # define PLATFORM_ID "BeOS"
00513
00514 #elif defined(__QNX__) || defined(__QNXNTO__)
00515 # define PLATFORM_ID "QNX"
```

```
00516
00517 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00518 # define PLATFORM_ID "Tru64"
00519
00520 #elif defined(__riscos) || defined(__riscos__)
00521 # define PLATFORM_ID "RISCos"
00522
00523 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00524 # define PLATFORM_ID "SINIX"
00525
00526 #elif defined(__UNIX_SV__)
00527 # define PLATFORM_ID "UNIX_SV"
00528
00529 #elif defined(__bsdos__)
00530 # define PLATFORM_ID "BSDOS"
00531
00532 #elif defined(_MPRAS) || defined(MPRAS)
00533 # define PLATFORM_ID "MP-RAS"
00534
00535 #elif defined(__osf) || defined(__osf__)
00536 # define PLATFORM_ID "OSF1"
00537
00538 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00539 # define PLATFORM_ID "SCO_SV"
00540
00541 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00542 # define PLATFORM_ID "ULTRIX"
00543
00544 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00545 # define PLATFORM_ID "Xenix"
00546
00547 #elif defined(__WATCOMC__)
00548 # if defined(__LINUX__)
00549 #  define PLATFORM_ID "Linux"
00550
00551 # elif defined(__DOS__)
00552 #  define PLATFORM_ID "DOS"
00553
00554 # elif defined(__OS2__)
00555 #  define PLATFORM_ID "OS2"
00556
00557 # elif defined(__WINDOWS__)
00558 #  define PLATFORM_ID "Windows3x"
00559
00560 # elif defined(__VXWORKS__)
00561 #  define PLATFORM_ID "VxWorks"
00562
00563 # else /* unknown platform */
00564 #  define PLATFORM_ID
00565 # endif
00566
00567 #elif defined(__INTEGRITY)
00568 # if defined(INT_178B)
00569 #  define PLATFORM_ID "Integrity178"
00570
00571 # else /* regular Integrity */
00572 #  define PLATFORM_ID "Integrity"
00573 # endif
00574
00575 # elif defined(_ADI_COMPILER)
00576 #  define PLATFORM_ID "ADSP"
00577
00578 #else /* unknown platform */
00579 # define PLATFORM_ID
00580
00581 #endif
00582
00583 /* For windows compilers MSVC and Intel we can determine
00584    the architecture of the compiler being used.  This is because
00585    the compilers do not have flags that can change the architecture,
00586    but rather depend on which compiler is being used
00587 */
00588 #if defined(_WIN32) && defined(_MSC_VER)
00589 # if defined(_M_IA64)
00590 #  define ARCHITECTURE_ID "IA64"
00591
00592 # elif defined(_M_ARM64EC)
00593 #  define ARCHITECTURE_ID "ARM64EC"
00594
00595 # elif defined(_M_X64) || defined(_M_AMD64)
00596 #  define ARCHITECTURE_ID "x64"
00597
00598 # elif defined(_M_IX86)
00599 #  define ARCHITECTURE_ID "X86"
00600
00601 # elif defined(_M_ARM64)
00602 #  define ARCHITECTURE_ID "ARM64"
```

```
00603
00604 # elif defined(_M_ARM)
00605 #  if _M_ARM == 4
00606 #   define ARCHITECTURE_ID "ARMV4I"
00607 #  elif _M_ARM == 5
00608 #   define ARCHITECTURE_ID "ARMV5I"
00609 #  else
00610 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00611 #  endif
00612
00613 # elif defined(_M_MIPS)
00614 #  define ARCHITECTURE_ID "MIPS"
00615
00616 # elif defined(_M_SH)
00617 #  define ARCHITECTURE_ID "SHx"
00618
00619 # else /* unknown architecture */
00620 #  define ARCHITECTURE_ID ""
00621 # endif
00622
00623 #elif defined(__WATCOMC__)
00624 # if defined(_M_I86)
00625 #  define ARCHITECTURE_ID "I86"
00626
00627 # elif defined(_M_IX86)
00628 #  define ARCHITECTURE_ID "X86"
00629
00630 # else /* unknown architecture */
00631 #  define ARCHITECTURE_ID ""
00632 # endif
00633
00634 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00635 # if defined(__ICCARM__)
00636 #  define ARCHITECTURE_ID "ARM"
00637
00638 # elif defined(__ICCRX__)
00639 #  define ARCHITECTURE_ID "RX"
00640
00641 # elif defined(__ICCRH850__)
00642 #  define ARCHITECTURE_ID "RH850"
00643
00644 # elif defined(__ICCRL78__)
00645 #  define ARCHITECTURE_ID "RL78"
00646
00647 # elif defined(__ICCRISCV__)
00648 #  define ARCHITECTURE_ID "RISCV"
00649
00650 # elif defined(__ICCAVR__)
00651 #  define ARCHITECTURE_ID "AVR"
00652
00653 # elif defined(__ICC430__)
00654 #  define ARCHITECTURE_ID "MSP430"
00655
00656 # elif defined(__ICCV850__)
00657 #  define ARCHITECTURE_ID "V850"
00658
00659 # elif defined(__ICC8051__)
00660 #  define ARCHITECTURE_ID "8051"
00661
00662 # elif defined(__ICCSTM8__)
00663 #  define ARCHITECTURE_ID "STM8"
00664
00665 # else /* unknown architecture */
00666 #  define ARCHITECTURE_ID ""
00667 # endif
00668
00669 #elif defined(__ghs__)
00670 # if defined(__PPC64__)
00671 #  define ARCHITECTURE_ID "PPC64"
00672
00673 # elif defined(__ppc__)
00674 #  define ARCHITECTURE_ID "PPC"
00675
00676 # elif defined(__ARM__)
00677 #  define ARCHITECTURE_ID "ARM"
00678
00679 # elif defined(__x86_64__)
00680 #  define ARCHITECTURE_ID "x64"
00681
00682 # elif defined(__i386__)
00683 #  define ARCHITECTURE_ID "X86"
00684
00685 # else /* unknown architecture */
00686 #  define ARCHITECTURE_ID ""
00687 # endif
00688
00689 #elif defined(__clang__) && defined(__ti__)
```

```
00690 # if defined(__ARM_ARCH)
00691 #  define ARCHITECTURE_ID "Arm"
00692
00693 # else /* unknown architecture */
00694 #  define ARCHITECTURE_ID ""
00695 # endif
00696
00697 #elif defined(__TI_COMPILER_VERSION__)
00698 # if defined(__TI_ARM__)
00699 #  define ARCHITECTURE_ID "ARM"
00700
00701 # elif defined(__MSP430__)
00702 #  define ARCHITECTURE_ID "MSP430"
00703
00704 # elif defined(__TMS320C28XX__)
00705 #  define ARCHITECTURE_ID "TMS320C28x"
00706
00707 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00708 #  define ARCHITECTURE_ID "TMS320C6x"
00709
00710 # else /* unknown architecture */
00711 #  define ARCHITECTURE_ID ""
00712 # endif
00713
00714 # elif defined(__ADSPSHARC__)
00715 #  define ARCHITECTURE_ID "SHARC"
00716
00717 # elif defined(__ADSPBLACKFIN__)
00718 #  define ARCHITECTURE_ID "Blackfin"
00719
00720 #elif defined(__TASKING__)
00721
00722 # if defined(__CTC__) || defined(__CPTC__)
00723 #  define ARCHITECTURE_ID "TriCore"
00724
00725 # elif defined(__CMCS__)
00726 #  define ARCHITECTURE_ID "MCS"
00727
00728 # elif defined(__CARM__)
00729 #  define ARCHITECTURE_ID "ARM"
00730
00731 # elif defined(__CARC__)
00732 #  define ARCHITECTURE_ID "ARC"
00733
00734 # elif defined(__C51__)
00735 #  define ARCHITECTURE_ID "8051"
00736
00737 # elif defined(__CPCP__)
00738 #  define ARCHITECTURE_ID "PCP"
00739
00740 # else
00741 #  define ARCHITECTURE_ID ""
00742 # endif
00743
00744 #else
00745 #  define ARCHITECTURE_ID
00746 #endif
00747
00748 /* Convert integer to decimal digit literals.  */
00749 #define DEC(n)                   \
00750   ('0' + (((n) / 10000000)%10)), \
00751   ('0' + (((n) / 1000000)%10)),  \
00752   ('0' + (((n) / 100000)%10)),   \
00753   ('0' + (((n) / 10000)%10)),    \
00754   ('0' + (((n) / 1000)%10)),     \
00755   ('0' + (((n) / 100)%10)),      \
00756   ('0' + (((n) / 10)%10)),       \
00757   ('0' +  ((n) % 10))
00758
00759 /* Convert integer to hex digit literals.  */
00760 #define HEX(n)             \
00761   ('0' + ((n)>>28 & 0xF)), \
00762   ('0' + ((n)>>24 & 0xF)), \
00763   ('0' + ((n)>>20 & 0xF)), \
00764   ('0' + ((n)>>16 & 0xF)), \
00765   ('0' + ((n)>>12 & 0xF)), \
00766   ('0' + ((n)>>8  & 0xF)), \
00767   ('0' + ((n)>>4  & 0xF)), \
00768   ('0' + ((n)     & 0xF))
00769
00770 /* Construct a string literal encoding the version number. */
00771 #ifdef COMPILER_VERSION
00772 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "]";
00773
00774 /* Construct a string literal encoding the version number components. */
00775 #elif defined(COMPILER_VERSION_MAJOR)
00776 char const info_version[] = {
```

```
00777   'I', 'N', 'F', 'O', ':',
00778   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00779   COMPILER_VERSION_MAJOR,
00780 # ifdef COMPILER_VERSION_MINOR
00781   '.', COMPILER_VERSION_MINOR,
00782 #  ifdef COMPILER_VERSION_PATCH
00783    '.', COMPILER_VERSION_PATCH,
00784 #   ifdef COMPILER_VERSION_TWEAK
00785     '.', COMPILER_VERSION_TWEAK,
00786 #   endif
00787 #  endif
00788 # endif
00789   ']','\0'};
00790 #endif
00791
00792 /* Construct a string literal encoding the internal version number. */
00793 #ifdef COMPILER_VERSION_INTERNAL
00794 char const info_version_internal[] = {
00795   'I', 'N', 'F', 'O', ':',
00796   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00797   'i','n','t','e','r','n','a','l','[',
00798   COMPILER_VERSION_INTERNAL,']','\0'};
00799 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00800 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
      COMPILER_VERSION_INTERNAL_STR "]";
00801 #endif
00802
00803 /* Construct a string literal encoding the version number components. */
00804 #ifdef SIMULATE_VERSION_MAJOR
00805 char const info_simulate_version[] = {
00806   'I', 'N', 'F', 'O', ':',
00807   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00808   SIMULATE_VERSION_MAJOR,
00809 # ifdef SIMULATE_VERSION_MINOR
00810   '.', SIMULATE_VERSION_MINOR,
00811 #  ifdef SIMULATE_VERSION_PATCH
00812    '.', SIMULATE_VERSION_PATCH,
00813 #   ifdef SIMULATE_VERSION_TWEAK
00814     '.', SIMULATE_VERSION_TWEAK,
00815 #   endif
00816 #  endif
00817 # endif
00818   ']','\0'};
00819 #endif
00820
00821 /* Construct the string literal in pieces to prevent the source from
00822    getting matched.  Store it in a pointer rather than an array
00823    because some compilers will just produce instructions to fill the
00824    array rather than assigning a pointer to a static array.  */
00825 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00826 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00827
00828
00829
00830 #define C_STD_99 199901L
00831 #define C_STD_11 201112L
00832 #define C_STD_17 201710L
00833 #define C_STD_23 202311L
00834
00835 #ifdef __STDC_VERSION__
00836 #  define C_STD __STDC_VERSION__
00837 #endif
00838
00839 #if !defined(__STDC__) && !defined(__clang__)
00840 # if defined(_MSC_VER) || defined(__ibmxl__) || defined(__IBMC__)
00841 #  define C_VERSION "90"
00842 # else
00843 #  define C_VERSION
00844 # endif
00845 #elif C_STD > C_STD_17
00846 # define C_VERSION "23"
00847 #elif C_STD > C_STD_11
00848 # define C_VERSION "17"
00849 #elif C_STD > C_STD_99
00850 # define C_VERSION "11"
00851 #elif C_STD >= C_STD_99
00852 # define C_VERSION "99"
00853 #else
00854 # define C_VERSION "90"
00855 #endif
00856 const char* info_language_standard_default =
00857   "INFO" ":" "standard_default[" C_VERSION "]";
00858
00859 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00860 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlC__) ||           \
00861      defined(__TI_COMPILER_VERSION__)) &&                                     \
00862   !defined(__STRICT_ANSI__)
```

```
00863   "ON"
00864 #else
00865   "OFF"
00866 #endif
00867 "]";
00868
00869 /*--------------------------------------------------------------------------*/
00870
00871 #ifdef ID_VOID_MAIN
00872 void main() {}
00873 #else
00874 # if defined(__CLASSIC_C__)
00875 int main(argc, argv) int argc; char *argv[];
00876 # else
00877 int main(int argc, char* argv[])
00878 # endif
00879 {
00880   int require = 0;
00881   require += info_compiler[argc];
00882   require += info_platform[argc];
00883   require += info_arch[argc];
00884 #ifdef COMPILER_VERSION_MAJOR
00885   require += info_version[argc];
00886 #endif
00887 #ifdef COMPILER_VERSION_INTERNAL
00888   require += info_version_internal[argc];
00889 #endif
00890 #ifdef SIMULATE_ID
00891   require += info_simulate[argc];
00892 #endif
00893 #ifdef SIMULATE_VERSION_MAJOR
00894   require += info_simulate_version[argc];
00895 #endif
00896 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00897   require += info_cray[argc];
00898 #endif
00899   require += info_language_standard_default[argc];
00900   require += info_language_extensions_default[argc];
00901   (void)argv;
00902   return require;
00903 }
00904 #endif
```

6.5 build/CMakeFiles/3.30.5/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x)
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X)
- #define STRINGIFY(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD_98 199711L
- #define CXX_STD_11 201103L
- #define CXX_STD_14 201402L
- #define CXX_STD_17 201703L
- #define CXX_STD_20 202002L
- #define CXX_STD_23 202302L
- #define CXX_STD __cplusplus

Functions

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

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

6.5.1 Macro Definition Documentation

6.5.1.1 __has_include

```
#define __has_include(
              x)
```

Value:

```
0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

6.5.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 724 of file CMakeCXXCompilerId.cpp.

6.5.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 427 of file CMakeCXXCompilerId.cpp.

6.5.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 861 of file CMakeCXXCompilerId.cpp.

6.5.1.5 CXX_STD_11

```
#define CXX_STD_11 201103L
```

Definition at line 810 of file CMakeCXXCompilerId.cpp.

6.5.1.6 CXX_STD_14

```
#define CXX_STD_14 201402L
```

Definition at line 811 of file CMakeCXXCompilerId.cpp.

6.5.1.7 CXX_STD_17

```
#define CXX_STD_17 201703L
```

Definition at line 812 of file CMakeCXXCompilerId.cpp.

6.5.1.8 CXX_STD_20

```
#define CXX_STD_20 202002L
```

Definition at line 813 of file CMakeCXXCompilerId.cpp.

6.5.1.9 CXX_STD_23

```
#define CXX_STD_23 202302L
```

Definition at line 814 of file CMakeCXXCompilerId.cpp.

6.5.1.10 CXX_STD_98

```
#define CXX_STD_98 199711L
```

Definition at line 809 of file CMakeCXXCompilerId.cpp.

6.5.1.11 DEC

```
#define DEC(
              n)
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 728 of file CMakeCXXCompilerId.cpp.

6.5.1.12 HEX

```
#define HEX(
               n)
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 739 of file CMakeCXXCompilerId.cpp.

6.5.1.13 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 558 of file CMakeCXXCompilerId.cpp.

6.5.1.14 STRINGIFY

```
#define STRINGIFY(
               X)
```

Value:

```
STRINGIFY_HELPER(X)
```

Definition at line 448 of file CMakeCXXCompilerId.cpp.

6.5.1.15 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
               X)
```

Value:

```
#X
```

Definition at line 447 of file CMakeCXXCompilerId.cpp.

6.5.2 Function Documentation

6.5.2.1 main()

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

Definition at line 894 of file CMakeCXXCompilerId.cpp.

6.5.3 Variable Documentation

6.5.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 805 of file CMakeCXXCompilerId.cpp.

6.5.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 434 of file CMakeCXXCompilerId.cpp.

6.5.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["



  "OFF"
"]"
```

Definition at line 882 of file CMakeCXXCompilerId.cpp.

6.5.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["









  "98"
"]"
```

Definition at line 864 of file CMakeCXXCompilerId.cpp.

6.5.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 804 of file CMakeCXXCompilerId.cpp.

6.6 CMakeCXXCompilerId.cpp

Go to the documentation of this file.

```
00001 /* This source file must have a .cpp extension so that all C++ compilers
00002    recognize the extension without flags.  Borland does not know .cxx for
00003    example.  */
00004 #ifndef __cplusplus
00005 # error "A C compiler has been selected for C++."
00006 #endif
00007
00008 #if !defined(__has_include)
00009 /* If the compiler does not have __has_include, pretend the answer is
00010    always no.  */
00011 #  define __has_include(x) 0
00012 #endif
00013
00014
00015 /* Version number components: V=Version, R=Revision, P=Patch
00016    Version date components:   YYYY=Year, MM=Month,   DD=Day  */
00017
00018 #if defined(__INTEL_COMPILER) || defined(__ICC)
00019 # define COMPILER_ID "Intel"
00020 # if defined(_MSC_VER)
00021 #  define SIMULATE_ID "MSVC"
00022 # endif
00023 # if defined(__GNUC__)
00024 #  define SIMULATE_ID "GNU"
00025 # endif
00026   /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00027      except that a few beta releases use the old format with V=2021.  */
00028 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00029 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00030 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00031 #  if defined(__INTEL_COMPILER_UPDATE)
00032 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00033 #  else
00034 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00035 #  endif
00036 # else
00037 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00038 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00039    /* The third version component from --version is an update index,
00040       but no macro is provided for it.  */
00041 #  define COMPILER_VERSION_PATCH DEC(0)
00042 # endif
00043 # if defined(__INTEL_COMPILER_BUILD_DATE)
00044    /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00045 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00046 # endif
00047 # if defined(_MSC_VER)
00048    /* _MSC_VER = VVRR */
00049 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00050 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00051 # endif
00052 # if defined(__GNUC__)
00053 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00054 # elif defined(__GNUG__)
00055 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00056 # endif
00057 # if defined(__GNUC_MINOR__)
00058 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00059 # endif
00060 # if defined(__GNUC_PATCHLEVEL__)
00061 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00062 # endif
00063
00064 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00065 # define COMPILER_ID "IntelLLVM"
00066 #if defined(_MSC_VER)
00067 # define SIMULATE_ID "MSVC"
00068 #endif
00069 #if defined(__GNUC__)
00070 # define SIMULATE_ID "GNU"
00071 #endif
00072 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00073  * later.  Look for 6 digit vs. 8 digit version number to decide encoding.
00074  * VVVV is no smaller than the current year when a version is released.
00075  */
00076 #if __INTEL_LLVM_COMPILER < 1000000L
00077 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00078 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00079 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER    % 10)
00080 #else
00081 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00082 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
```

```
00083 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER     % 100)
00084 #endif
00085 #if defined(_MSC_VER)
00086   /* _MSC_VER = VVRR */
00087 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00088 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00089 #endif
00090 #if defined(__GNUC__)
00091 # define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00092 #elif defined(__GNUG__)
00093 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00094 #endif
00095 #if defined(__GNUC_MINOR__)
00096 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00097 #endif
00098 #if defined(__GNUC_PATCHLEVEL__)
00099 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00100 #endif
00101
00102 #elif defined(__PATHCC__)
00103 # define COMPILER_ID "PathScale"
00104 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00105 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00106 # if defined(__PATHCC_PATCHLEVEL__)
00107 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00108 # endif
00109
00110 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00111 # define COMPILER_ID "Embarcadero"
00112 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00113 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00114 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00115
00116 #elif defined(__BORLANDC__)
00117 # define COMPILER_ID "Borland"
00118   /* __BORLANDC__ = 0xVRR */
00119 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00120 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00121
00122 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00123 # define COMPILER_ID "Watcom"
00124   /* __WATCOMC__ = VVRR */
00125 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00126 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00127 # if (__WATCOMC__ % 10) > 0
00128 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00129 # endif
00130
00131 #elif defined(__WATCOMC__)
00132 # define COMPILER_ID "OpenWatcom"
00133   /* __WATCOMC__ = VVRP + 1100 */
00134 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00135 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00136 # if (__WATCOMC__ % 10) > 0
00137 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00138 # endif
00139
00140 #elif defined(__SUNPRO_CC)
00141 # define COMPILER_ID "SunPro"
00142 # if __SUNPRO_CC >= 0x5100
00143   /* __SUNPRO_CC = 0xVRRP */
00144 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»12)
00145 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xFF)
00146 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00147 # else
00148   /* __SUNPRO_CC = 0xVRP */
00149 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»8)
00150 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xF)
00151 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00152 # endif
00153
00154 #elif defined(__HP_aCC)
00155 # define COMPILER_ID "HP"
00156   /* __HP_aCC = VVRRPP */
00157 # define COMPILER_VERSION_MAJOR DEC(__HP_aCC/10000)
00158 # define COMPILER_VERSION_MINOR DEC(__HP_aCC/100 % 100)
00159 # define COMPILER_VERSION_PATCH DEC(__HP_aCC     % 100)
00160
00161 #elif defined(__DECCXX)
00162 # define COMPILER_ID "Compaq"
00163   /* __DECCXX_VER = VVRRTPPPP */
00164 # define COMPILER_VERSION_MAJOR DEC(__DECCXX_VER/10000000)
00165 # define COMPILER_VERSION_MINOR DEC(__DECCXX_VER/100000  % 100)
00166 # define COMPILER_VERSION_PATCH DEC(__DECCXX_VER         % 10000)
00167
00168 #elif defined(__IBMCPP__) && defined(__COMPILER_VER__)
00169 # define COMPILER_ID "zOS"
```

```
00170   /* __IBMCPP__ = VRP */
00171 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00172 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00173 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00174
00175 #elif defined(__open_xl__) && defined(__clang__)
00176 # define COMPILER_ID "IBMClang"
00177 # define COMPILER_VERSION_MAJOR DEC(__open_xl_version__)
00178 # define COMPILER_VERSION_MINOR DEC(__open_xl_release__)
00179 # define COMPILER_VERSION_PATCH DEC(__open_xl_modification__)
00180 # define COMPILER_VERSION_TWEAK DEC(__open_xl_ptf_fix_level__)
00181
00182
00183 #elif defined(__ibmxl__) && defined(__clang__)
00184 # define COMPILER_ID "XLClang"
00185 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00186 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00187 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00188 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00189
00190
00191 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ >= 800
00192 # define COMPILER_ID "XL"
00193   /* __IBMCPP__ = VRP */
00194 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00195 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00196 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00197
00198 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ < 800
00199 # define COMPILER_ID "VisualAge"
00200   /* __IBMCPP__ = VRP */
00201 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00202 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00203 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00204
00205 #elif defined(__NVCOMPILER)
00206 # define COMPILER_ID "NVHPC"
00207 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00208 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00209 # if defined(__NVCOMPILER_PATCHLEVEL__)
00210 #  define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00211 # endif
00212
00213 #elif defined(__PGI)
00214 # define COMPILER_ID "PGI"
00215 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00216 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00217 # if defined(__PGIC_PATCHLEVEL__)
00218 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00219 # endif
00220
00221 #elif defined(__clang__) && defined(__cray__)
00222 # define COMPILER_ID "CrayClang"
00223 # define COMPILER_VERSION_MAJOR DEC(__cray_major__)
00224 # define COMPILER_VERSION_MINOR DEC(__cray_minor__)
00225 # define COMPILER_VERSION_PATCH DEC(__cray_patchlevel__)
00226 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00227
00228
00229 #elif defined(_CRAYC)
00230 # define COMPILER_ID "Cray"
00231 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00232 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00233
00234 #elif defined(__TI_COMPILER_VERSION__)
00235 # define COMPILER_ID "TI"
00236   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00237 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00238 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00239 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00240
00241 #elif defined(__CLANG_FUJITSU)
00242 # define COMPILER_ID "FujitsuClang"
00243 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00244 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00245 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00246 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00247
00248
00249 #elif defined(__FUJITSU)
00250 # define COMPILER_ID "Fujitsu"
00251 # if defined(__FCC_version__)
00252 #   define COMPILER_VERSION __FCC_version__
00253 # elif defined(__FCC_major__)
00254 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00255 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00256 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
```

```
00257 # endif
00258 # if defined(__fcc_version)
00259 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00260 # elif defined(__FCC_VERSION)
00261 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00262 # endif
00263
00264
00265 #elif defined(__ghs__)
00266 # define COMPILER_ID "GHS"
00267 /* __GHS_VERSION_NUMBER = VVVVRP */
00268 # ifdef __GHS_VERSION_NUMBER
00269 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00270 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00271 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00272 # endif
00273
00274 #elif defined(__TASKING__)
00275 # define COMPILER_ID "Tasking"
00276   # define COMPILER_VERSION_MAJOR DEC(__VERSION__/1000)
00277   # define COMPILER_VERSION_MINOR DEC(__VERSION__ % 100)
00278 # define COMPILER_VERSION_INTERNAL DEC(__VERSION__)
00279
00280 #elif defined(__ORANGEC__)
00281 # define COMPILER_ID "OrangeC"
00282 # define COMPILER_VERSION_MAJOR DEC(__ORANGEC_MAJOR__)
00283 # define COMPILER_VERSION_MINOR DEC(__ORANGEC_MINOR__)
00284 # define COMPILER_VERSION_PATCH DEC(__ORANGEC_PATCHLEVEL__)
00285
00286 #elif defined(__SCO_VERSION__)
00287 # define COMPILER_ID "SCO"
00288
00289 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00290 # define COMPILER_ID "ARMCC"
00291 #if __ARMCC_VERSION >= 1000000
00292   /* __ARMCC_VERSION = VRRPPPP */
00293   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00294   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00295   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00296 #else
00297   /* __ARMCC_VERSION = VRPPPP */
00298   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00299   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00300   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00301 #endif
00302
00303
00304 #elif defined(__clang__) && defined(__apple_build_version__)
00305 # define COMPILER_ID "AppleClang"
00306 # if defined(_MSC_VER)
00307 #  define SIMULATE_ID "MSVC"
00308 # endif
00309 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00310 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00311 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00312 # if defined(_MSC_VER)
00313   /* _MSC_VER = VVRR */
00314 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00315 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00316 # endif
00317 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00318
00319 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00320 # define COMPILER_ID "ARMClang"
00321   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00322   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00323   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION/100   % 100)
00324 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00325
00326 #elif defined(__clang__) && defined(__ti__)
00327 # define COMPILER_ID "TIClang"
00328   # define COMPILER_VERSION_MAJOR DEC(__ti_major__)
00329   # define COMPILER_VERSION_MINOR DEC(__ti_minor__)
00330   # define COMPILER_VERSION_PATCH DEC(__ti_patchlevel__)
00331 # define COMPILER_VERSION_INTERNAL DEC(__ti_version__)
00332
00333 #elif defined(__clang__)
00334 # define COMPILER_ID "Clang"
00335 # if defined(_MSC_VER)
00336 #  define SIMULATE_ID "MSVC"
00337 # endif
00338 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00339 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00340 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00341 # if defined(_MSC_VER)
00342   /* _MSC_VER = VVRR */
00343 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
```

```
00344 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00345 # endif
00346
00347 #elif defined(__LCC__) && (defined(__GNUC__) || defined(__GNUG__) || defined(__MCST__))
00348 # define COMPILER_ID "LCC"
00349 # define COMPILER_VERSION_MAJOR DEC(__LCC__ / 100)
00350 # define COMPILER_VERSION_MINOR DEC(__LCC__ % 100)
00351 # if defined(__LCC_MINOR__)
00352 #  define COMPILER_VERSION_PATCH DEC(__LCC_MINOR__)
00353 # endif
00354 # if defined(__GNUC__) && defined(__GNUC_MINOR__)
00355 #  define SIMULATE_ID "GNU"
00356 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00357 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00358 #  if defined(__GNUC_PATCHLEVEL__)
00359 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00360 #  endif
00361 # endif
00362
00363 #elif defined(__GNUC__) || defined(__GNUG__)
00364 # define COMPILER_ID "GNU"
00365 # if defined(__GNUC__)
00366 #  define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00367 # else
00368 #  define COMPILER_VERSION_MAJOR DEC(__GNUG__)
00369 # endif
00370 # if defined(__GNUC_MINOR__)
00371 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00372 # endif
00373 # if defined(__GNUC_PATCHLEVEL__)
00374 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00375 # endif
00376
00377 #elif defined(_MSC_VER)
00378 # define COMPILER_ID "MSVC"
00379   /* _MSC_VER = VVRR */
00380 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00381 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00382 # if defined(_MSC_FULL_VER)
00383 #  if _MSC_VER >= 1400
00384     /* _MSC_FULL_VER = VVRRPPPPP */
00385 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00386 #  else
00387     /* _MSC_FULL_VER = VVRRPPPP */
00388 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00389 #  endif
00390 # endif
00391 # if defined(_MSC_BUILD)
00392 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00393 # endif
00394
00395 #elif defined(_ADI_COMPILER)
00396 # define COMPILER_ID "ADSP"
00397 #if defined(__VERSIONNUM__)
00398   /* __VERSIONNUM__ = 0xVVRRPTT */
00399 #  define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ >> 24 & 0xFF)
00400 #  define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ >> 16 & 0xFF)
00401 #  define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ >> 8 & 0xFF)
00402 #  define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00403 #endif
00404
00405 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00406 # define COMPILER_ID "IAR"
00407 # if defined(__VER__) && defined(__ICCARM__)
00408 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00409 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00410 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00411 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00412 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
      defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
      defined(__ICC8051__) || defined(__ICCSTM8__))
00413 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00414 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00415 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00416 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00417 # endif
00418
00419
00420 /* These compilers are either not known or too old to define an
00421   identification macro.  Try to identify the platform and guess that
00422   it is the native compiler.  */
00423 #elif defined(__hpux) || defined(__hpua)
00424 # define COMPILER_ID "HP"
00425
00426 #else /* unknown compiler */
00427 # define COMPILER_ID ""
00428 #endif
```

```
00429
00430 /* Construct the string literal in pieces to prevent the source from
00431    getting matched.  Store it in a pointer rather than an array
00432    because some compilers will just produce instructions to fill the
00433    array rather than assigning a pointer to a static array.  */
00434 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00435 #ifdef SIMULATE_ID
00436 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00437 #endif
00438
00439 #ifdef __QNXNTO__
00440 char const* qnxnto = "INFO" ":" "qnxnto[]";
00441 #endif
00442
00443 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00444 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00445 #endif
00446
00447 #define STRINGIFY_HELPER(X) #X
00448 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00449
00450 /* Identify known platforms by name.  */
00451 #if defined(__linux) || defined(__linux__) || defined(linux)
00452 # define PLATFORM_ID "Linux"
00453
00454 #elif defined(__MSYS__)
00455 # define PLATFORM_ID "MSYS"
00456
00457 #elif defined(__CYGWIN__)
00458 # define PLATFORM_ID "Cygwin"
00459
00460 #elif defined(__MINGW32__)
00461 # define PLATFORM_ID "MinGW"
00462
00463 #elif defined(__APPLE__)
00464 # define PLATFORM_ID "Darwin"
00465
00466 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00467 # define PLATFORM_ID "Windows"
00468
00469 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00470 # define PLATFORM_ID "FreeBSD"
00471
00472 #elif defined(__NetBSD__) || defined(__NetBSD)
00473 # define PLATFORM_ID "NetBSD"
00474
00475 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00476 # define PLATFORM_ID "OpenBSD"
00477
00478 #elif defined(__sun) || defined(sun)
00479 # define PLATFORM_ID "SunOS"
00480
00481 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00482 # define PLATFORM_ID "AIX"
00483
00484 #elif defined(__hpux) || defined(__hpux__)
00485 # define PLATFORM_ID "HP-UX"
00486
00487 #elif defined(__HAIKU__)
00488 # define PLATFORM_ID "Haiku"
00489
00490 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00491 # define PLATFORM_ID "BeOS"
00492
00493 #elif defined(__QNX__) || defined(__QNXNTO__)
00494 # define PLATFORM_ID "QNX"
00495
00496 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00497 # define PLATFORM_ID "Tru64"
00498
00499 #elif defined(__riscos) || defined(__riscos__)
00500 # define PLATFORM_ID "RISCos"
00501
00502 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00503 # define PLATFORM_ID "SINIX"
00504
00505 #elif defined(__UNIX_SV__)
00506 # define PLATFORM_ID "UNIX_SV"
00507
00508 #elif defined(__bsdos__)
00509 # define PLATFORM_ID "BSDOS"
00510
00511 #elif defined(_MPRAS) || defined(MPRAS)
00512 # define PLATFORM_ID "MP-RAS"
00513
00514 #elif defined(__osf) || defined(__osf__)
00515 # define PLATFORM_ID "OSF1"
```

```
00516
00517 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00518 # define PLATFORM_ID "SCO_SV"
00519
00520 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00521 # define PLATFORM_ID "ULTRIX"
00522
00523 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00524 # define PLATFORM_ID "Xenix"
00525
00526 #elif defined(__WATCOMC__)
00527 # if defined(__LINUX__)
00528 #  define PLATFORM_ID "Linux"
00529
00530 # elif defined(__DOS__)
00531 #  define PLATFORM_ID "DOS"
00532
00533 # elif defined(__OS2__)
00534 #  define PLATFORM_ID "OS2"
00535
00536 # elif defined(__WINDOWS__)
00537 #  define PLATFORM_ID "Windows3x"
00538
00539 # elif defined(__VXWORKS__)
00540 #  define PLATFORM_ID "VxWorks"
00541
00542 # else /* unknown platform */
00543 #  define PLATFORM_ID
00544 # endif
00545
00546 #elif defined(__INTEGRITY)
00547 # if defined(INT_178B)
00548 #  define PLATFORM_ID "Integrity178"
00549
00550 # else /* regular Integrity */
00551 #  define PLATFORM_ID "Integrity"
00552 # endif
00553
00554 # elif defined(_ADI_COMPILER)
00555 #  define PLATFORM_ID "ADSP"
00556
00557 #else /* unknown platform */
00558 # define PLATFORM_ID
00559
00560 #endif
00561
00562 /* For windows compilers MSVC and Intel we can determine
00563    the architecture of the compiler being used.  This is because
00564    the compilers do not have flags that can change the architecture,
00565    but rather depend on which compiler is being used
00566 */
00567 #if defined(_WIN32) && defined(_MSC_VER)
00568 # if defined(_M_IA64)
00569 #  define ARCHITECTURE_ID "IA64"
00570
00571 # elif defined(_M_ARM64EC)
00572 #  define ARCHITECTURE_ID "ARM64EC"
00573
00574 # elif defined(_M_X64) || defined(_M_AMD64)
00575 #  define ARCHITECTURE_ID "x64"
00576
00577 # elif defined(_M_IX86)
00578 #  define ARCHITECTURE_ID "X86"
00579
00580 # elif defined(_M_ARM64)
00581 #  define ARCHITECTURE_ID "ARM64"
00582
00583 # elif defined(_M_ARM)
00584 #  if _M_ARM == 4
00585 #   define ARCHITECTURE_ID "ARMV4I"
00586 #  elif _M_ARM == 5
00587 #   define ARCHITECTURE_ID "ARMV5I"
00588 #  else
00589 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00590 #  endif
00591
00592 # elif defined(_M_MIPS)
00593 #  define ARCHITECTURE_ID "MIPS"
00594
00595 # elif defined(_M_SH)
00596 #  define ARCHITECTURE_ID "SHx"
00597
00598 # else /* unknown architecture */
00599 #  define ARCHITECTURE_ID ""
00600 # endif
00601
00602 #elif defined(__WATCOMC__)
```

```
00603 # if defined(_M_I86)
00604 #  define ARCHITECTURE_ID "I86"
00605
00606 # elif defined(_M_IX86)
00607 #  define ARCHITECTURE_ID "X86"
00608
00609 # else /* unknown architecture */
00610 #  define ARCHITECTURE_ID ""
00611 # endif
00612
00613 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00614 # if defined(__ICCARM__)
00615 #  define ARCHITECTURE_ID "ARM"
00616
00617 # elif defined(__ICCRX__)
00618 #  define ARCHITECTURE_ID "RX"
00619
00620 # elif defined(__ICCRH850__)
00621 #  define ARCHITECTURE_ID "RH850"
00622
00623 # elif defined(__ICCRL78__)
00624 #  define ARCHITECTURE_ID "RL78"
00625
00626 # elif defined(__ICCRISCV__)
00627 #  define ARCHITECTURE_ID "RISCV"
00628
00629 # elif defined(__ICCAVR__)
00630 #  define ARCHITECTURE_ID "AVR"
00631
00632 # elif defined(__ICC430__)
00633 #  define ARCHITECTURE_ID "MSP430"
00634
00635 # elif defined(__ICCV850__)
00636 #  define ARCHITECTURE_ID "V850"
00637
00638 # elif defined(__ICC8051__)
00639 #  define ARCHITECTURE_ID "8051"
00640
00641 # elif defined(__ICCSTM8__)
00642 #  define ARCHITECTURE_ID "STM8"
00643
00644 # else /* unknown architecture */
00645 #  define ARCHITECTURE_ID ""
00646 # endif
00647
00648 #elif defined(__ghs__)
00649 # if defined(__PPC64__)
00650 #  define ARCHITECTURE_ID "PPC64"
00651
00652 # elif defined(__ppc__)
00653 #  define ARCHITECTURE_ID "PPC"
00654
00655 # elif defined(__ARM__)
00656 #  define ARCHITECTURE_ID "ARM"
00657
00658 # elif defined(__x86_64__)
00659 #  define ARCHITECTURE_ID "x64"
00660
00661 # elif defined(__i386__)
00662 #  define ARCHITECTURE_ID "X86"
00663
00664 # else /* unknown architecture */
00665 #  define ARCHITECTURE_ID ""
00666 # endif
00667
00668 #elif defined(__clang__) && defined(__ti__)
00669 # if defined(__ARM_ARCH)
00670 #  define ARCHITECTURE_ID "Arm"
00671
00672 # else /* unknown architecture */
00673 #  define ARCHITECTURE_ID ""
00674 # endif
00675
00676 #elif defined(__TI_COMPILER_VERSION__)
00677 # if defined(__TI_ARM__)
00678 #  define ARCHITECTURE_ID "ARM"
00679
00680 # elif defined(__MSP430__)
00681 #  define ARCHITECTURE_ID "MSP430"
00682
00683 # elif defined(__TMS320C28XX__)
00684 #  define ARCHITECTURE_ID "TMS320C28x"
00685
00686 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00687 #  define ARCHITECTURE_ID "TMS320C6x"
00688
00689 # else /* unknown architecture */
```

```
00690 #  define ARCHITECTURE_ID ""
00691 # endif
00692
00693 # elif defined(__ADSPSHARC__)
00694 #  define ARCHITECTURE_ID "SHARC"
00695
00696 # elif defined(__ADSPBLACKFIN__)
00697 #  define ARCHITECTURE_ID "Blackfin"
00698
00699 #elif defined(__TASKING__)
00700
00701 # if defined(__CTC__) || defined(__CPTC__)
00702 #  define ARCHITECTURE_ID "TriCore"
00703
00704 # elif defined(__CMCS__)
00705 #  define ARCHITECTURE_ID "MCS"
00706
00707 # elif defined(__CARM__)
00708 #  define ARCHITECTURE_ID "ARM"
00709
00710 # elif defined(__CARC__)
00711 #  define ARCHITECTURE_ID "ARC"
00712
00713 # elif defined(__C51__)
00714 #  define ARCHITECTURE_ID "8051"
00715
00716 # elif defined(__CPCP__)
00717 #  define ARCHITECTURE_ID "PCP"
00718
00719 # else
00720 #  define ARCHITECTURE_ID ""
00721 # endif
00722
00723 #else
00724 #  define ARCHITECTURE_ID
00725 #endif
00726
00727 /* Convert integer to decimal digit literals.  */
00728 #define DEC(n)                   \
00729   ('0' + (((n) / 10000000)%10)), \
00730   ('0' + (((n) / 1000000)%10)),  \
00731   ('0' + (((n) / 100000)%10)),   \
00732   ('0' + (((n) / 10000)%10)),    \
00733   ('0' + (((n) / 1000)%10)),     \
00734   ('0' + (((n) / 100)%10)),      \
00735   ('0' + (((n) / 10)%10)),       \
00736   ('0' +  ((n) % 10))
00737
00738 /* Convert integer to hex digit literals.  */
00739 #define HEX(n)             \
00740   ('0' + ((n)>>28 & 0xF)), \
00741   ('0' + ((n)>>24 & 0xF)), \
00742   ('0' + ((n)>>20 & 0xF)), \
00743   ('0' + ((n)>>16 & 0xF)), \
00744   ('0' + ((n)>>12 & 0xF)), \
00745   ('0' + ((n)>>8  & 0xF)), \
00746   ('0' + ((n)>>4  & 0xF)), \
00747   ('0' + ((n)     & 0xF))
00748
00749 /* Construct a string literal encoding the version number. */
00750 #ifdef COMPILER_VERSION
00751 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "]";
00752
00753 /* Construct a string literal encoding the version number components. */
00754 #elif defined(COMPILER_VERSION_MAJOR)
00755 char const info_version[] = {
00756   'I', 'N', 'F', 'O', ':',
00757   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00758   COMPILER_VERSION_MAJOR,
00759 # ifdef COMPILER_VERSION_MINOR
00760   '.', COMPILER_VERSION_MINOR,
00761 #  ifdef COMPILER_VERSION_PATCH
00762   '.', COMPILER_VERSION_PATCH,
00763 #   ifdef COMPILER_VERSION_TWEAK
00764     '.', COMPILER_VERSION_TWEAK,
00765 #   endif
00766 #  endif
00767 # endif
00768   ']','\0'};
00769 #endif
00770
00771 /* Construct a string literal encoding the internal version number. */
00772 #ifdef COMPILER_VERSION_INTERNAL
00773 char const info_version_internal[] = {
00774   'I', 'N', 'F', 'O', ':',
00775   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00776   'i','n','t','e','r','n','a','l','[',
```

```
00777   COMPILER_VERSION_INTERNAL,']','\0'};
00778 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00779 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
      COMPILER_VERSION_INTERNAL_STR "]";
00780 #endif
00781
00782 /* Construct a string literal encoding the version number components. */
00783 #ifdef SIMULATE_VERSION_MAJOR
00784 char const info_simulate_version[] = {
00785   'I', 'N', 'F', 'O', ':',
00786   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00787   SIMULATE_VERSION_MAJOR,
00788 # ifdef SIMULATE_VERSION_MINOR
00789   '.', SIMULATE_VERSION_MINOR,
00790 #  ifdef SIMULATE_VERSION_PATCH
00791    '.', SIMULATE_VERSION_PATCH,
00792 #   ifdef SIMULATE_VERSION_TWEAK
00793     '.', SIMULATE_VERSION_TWEAK,
00794 #   endif
00795 #  endif
00796 # endif
00797   ']','\0'};
00798 #endif
00799
00800 /* Construct the string literal in pieces to prevent the source from
00801    getting matched.  Store it in a pointer rather than an array
00802    because some compilers will just produce instructions to fill the
00803    array rather than assigning a pointer to a static array.  */
00804 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00805 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00806
00807
00808
00809 #define CXX_STD_98 199711L
00810 #define CXX_STD_11 201103L
00811 #define CXX_STD_14 201402L
00812 #define CXX_STD_17 201703L
00813 #define CXX_STD_20 202002L
00814 #define CXX_STD_23 202302L
00815
00816 #if defined(__INTEL_COMPILER) && defined(_MSVC_LANG)
00817 #  if _MSVC_LANG > CXX_STD_17
00818 #    define CXX_STD _MSVC_LANG
00819 #  elif _MSVC_LANG == CXX_STD_17 && defined(__cpp_aggregate_paren_init)
00820 #    define CXX_STD CXX_STD_20
00821 #  elif _MSVC_LANG > CXX_STD_14 && __cplusplus > CXX_STD_17
00822 #    define CXX_STD CXX_STD_20
00823 #  elif _MSVC_LANG > CXX_STD_14
00824 #    define CXX_STD CXX_STD_17
00825 #  elif defined(__INTEL_CXX11_MODE__) && defined(__cpp_aggregate_nsdmi)
00826 #    define CXX_STD CXX_STD_14
00827 #  elif defined(__INTEL_CXX11_MODE__)
00828 #    define CXX_STD CXX_STD_11
00829 #  else
00830 #    define CXX_STD CXX_STD_98
00831 #  endif
00832 #elif defined(_MSC_VER) && defined(_MSVC_LANG)
00833 #  if _MSVC_LANG > __cplusplus
00834 #    define CXX_STD _MSVC_LANG
00835 #  else
00836 #    define CXX_STD __cplusplus
00837 #  endif
00838 #elif defined(__NVCOMPILER)
00839 #  if __cplusplus == CXX_STD_17 && defined(__cpp_aggregate_paren_init)
00840 #    define CXX_STD CXX_STD_20
00841 #  else
00842 #    define CXX_STD __cplusplus
00843 #  endif
00844 #elif defined(__INTEL_COMPILER) || defined(__PGI)
00845 #  if __cplusplus == CXX_STD_11 && defined(__cpp_namespace_attributes)
00846 #    define CXX_STD CXX_STD_17
00847 #  elif __cplusplus == CXX_STD_11 && defined(__cpp_aggregate_nsdmi)
00848 #    define CXX_STD CXX_STD_14
00849 #  else
00850 #    define CXX_STD __cplusplus
00851 #  endif
00852 #elif (defined(__IBMCPP__) || defined(__ibmxl__)) && defined(__linux__)
00853 #  if __cplusplus == CXX_STD_11 && defined(__cpp_aggregate_nsdmi)
00854 #    define CXX_STD CXX_STD_14
00855 #  else
00856 #    define CXX_STD __cplusplus
00857 #  endif
00858 #elif __cplusplus == 1 && defined(__GXX_EXPERIMENTAL_CXX0X__)
00859 #  define CXX_STD CXX_STD_11
00860 #else
00861 #  define CXX_STD __cplusplus
00862 #endif
```

```
00863
00864 const char* info_language_standard_default = "INFO" ":" "standard_default["
00865 #if CXX_STD > CXX_STD_23
00866   "26"
00867 #elif CXX_STD > CXX_STD_20
00868   "23"
00869 #elif CXX_STD > CXX_STD_17
00870   "20"
00871 #elif CXX_STD > CXX_STD_14
00872   "17"
00873 #elif CXX_STD > CXX_STD_11
00874   "14"
00875 #elif CXX_STD >= CXX_STD_11
00876   "11"
00877 #else
00878   "98"
00879 #endif
00880 "]";
00881
00882 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00883 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlC__) ||           \
00884      defined(__TI_COMPILER_VERSION__)) &&                                     \
00885   !defined(__STRICT_ANSI__)
00886   "ON"
00887 #else
00888   "OFF"
00889 #endif
00890 "]";
00891
00892 /*--------------------------------------------------------------------------*/
00893
00894 int main(int argc, char* argv[])
00895 {
00896   int require = 0;
00897   require += info_compiler[argc];
00898   require += info_platform[argc];
00899   require += info_arch[argc];
00900 #ifdef COMPILER_VERSION_MAJOR
00901   require += info_version[argc];
00902 #endif
00903 #ifdef COMPILER_VERSION_INTERNAL
00904   require += info_version_internal[argc];
00905 #endif
00906 #ifdef SIMULATE_ID
00907   require += info_simulate[argc];
00908 #endif
00909 #ifdef SIMULATE_VERSION_MAJOR
00910   require += info_simulate_version[argc];
00911 #endif
00912 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00913   require += info_cray[argc];
00914 #endif
00915   require += info_language_standard_default[argc];
00916   require += info_language_extensions_default[argc];
00917   (void)argv;
00918   return require;
00919 }
```

6.7 build/CMakeFiles/3.31.0/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x)
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X)
- #define STRINGIFY(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD_98 199711L
- #define CXX_STD_11 201103L

- #define CXX_STD_14 201402L
- #define CXX_STD_17 201703L
- #define CXX_STD_20 202002L
- #define CXX_STD_23 202302L
- #define CXX_STD __cplusplus

Functions

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

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

6.7.1 Macro Definition Documentation

6.7.1.1 __has_include

```
#define __has_include(
              x)
```

Value:

```
0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

6.7.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 724 of file CMakeCXXCompilerId.cpp.

6.7.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 427 of file CMakeCXXCompilerId.cpp.

6.7.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 861 of file CMakeCXXCompilerId.cpp.

6.7.1.5 CXX_STD_11

```
#define CXX_STD_11 201103L
```

Definition at line 810 of file CMakeCXXCompilerId.cpp.

6.7.1.6 CXX_STD_14

```
#define CXX_STD_14 201402L
```

Definition at line 811 of file CMakeCXXCompilerId.cpp.

6.7.1.7 CXX_STD_17

```
#define CXX_STD_17 201703L
```

Definition at line 812 of file CMakeCXXCompilerId.cpp.

6.7.1.8 CXX_STD_20

```
#define CXX_STD_20 202002L
```

Definition at line 813 of file CMakeCXXCompilerId.cpp.

6.7.1.9 CXX_STD_23

```
#define CXX_STD_23 202302L
```

Definition at line 814 of file CMakeCXXCompilerId.cpp.

6.7.1.10 CXX_STD_98

```
#define CXX_STD_98 199711L
```

Definition at line 809 of file CMakeCXXCompilerId.cpp.

6.7.1.11 DEC

```
#define DEC(
              n)
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 728 of file CMakeCXXCompilerId.cpp.

6.7.1.12 HEX

```
#define HEX(
              n)
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 739 of file CMakeCXXCompilerId.cpp.

6.7.1.13 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 558 of file CMakeCXXCompilerId.cpp.

6.7.1.14 STRINGIFY

```
#define STRINGIFY(
              X)
```

Value:

```
STRINGIFY_HELPER(X)
```

Definition at line 448 of file CMakeCXXCompilerId.cpp.

6.7.1.15 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X)
```

Value:

```
#X
```

Definition at line 447 of file CMakeCXXCompilerId.cpp.

6.7.2 Function Documentation

6.7.2.1 main()

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

Definition at line 894 of file CMakeCXXCompilerId.cpp.

6.7.3 Variable Documentation

6.7.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 805 of file CMakeCXXCompilerId.cpp.

6.7.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 434 of file CMakeCXXCompilerId.cpp.

6.7.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["


  "OFF"
"]"
```

Definition at line 882 of file CMakeCXXCompilerId.cpp.

6.7.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["


  "98"
"]"
```

Definition at line 864 of file CMakeCXXCompilerId.cpp.

6.7.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 804 of file CMakeCXXCompilerId.cpp.

6.8 CMakeCXXCompilerId.cpp

Go to the documentation of this file.

```
00001 /* This source file must have a .cpp extension so that all C++ compilers
00002    recognize the extension without flags.  Borland does not know .cxx for
00003    example.  */
00004 #ifndef __cplusplus
00005 # error "A C compiler has been selected for C++."
00006 #endif
00007
00008 #if !defined(__has_include)
00009 /* If the compiler does not have __has_include, pretend the answer is
00010    always no.  */
00011 #  define __has_include(x) 0
00012 #endif
00013
00014
00015 /* Version number components: V=Version, R=Revision, P=Patch
00016    Version date components:   YYYY=Year, MM=Month,   DD=Day  */
00017
00018 #if defined(__INTEL_COMPILER) || defined(__ICC)
00019 # define COMPILER_ID "Intel"
00020 # if defined(_MSC_VER)
00021 #  define SIMULATE_ID "MSVC"
00022 # endif
00023 # if defined(__GNUC__)
00024 #  define SIMULATE_ID "GNU"
00025 # endif
00026   /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00027      except that a few beta releases use the old format with V=2021.  */
00028 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00029 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00030 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00031 #  if defined(__INTEL_COMPILER_UPDATE)
00032 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00033 #  else
00034 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00035 #  endif
00036 # else
00037 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00038 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00039    /* The third version component from --version is an update index,
00040       but no macro is provided for it.  */
00041 #  define COMPILER_VERSION_PATCH DEC(0)
00042 # endif
00043 # if defined(__INTEL_COMPILER_BUILD_DATE)
00044    /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00045 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00046 # endif
00047 # if defined(_MSC_VER)
00048    /* _MSC_VER = VVRR */
00049 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00050 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00051 # endif
00052 # if defined(__GNUC__)
00053 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00054 # elif defined(__GNUG__)
00055 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00056 # endif
00057 # if defined(__GNUC_MINOR__)
00058 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00059 # endif
00060 # if defined(__GNUC_PATCHLEVEL__)
00061 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00062 # endif
00063
00064 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00065 # define COMPILER_ID "IntelLLVM"
00066 #if defined(_MSC_VER)
00067 # define SIMULATE_ID "MSVC"
00068 #endif
00069 #if defined(__GNUC__)
00070 # define SIMULATE_ID "GNU"
00071 #endif
00072 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00073  * later.  Look for 6 digit vs. 8 digit version number to decide encoding.
00074  * VVVV is no smaller than the current year when a version is released.
00075  */
00076 #if __INTEL_LLVM_COMPILER < 1000000L
00077 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00078 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00079 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER    % 10)
00080 #else
00081 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00082 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
```

```
00083 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER     % 100)
00084 #endif
00085 #if defined(_MSC_VER)
00086   /* _MSC_VER = VVRR */
00087 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00088 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00089 #endif
00090 #if defined(__GNUC__)
00091 # define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00092 #elif defined(__GNUG__)
00093 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00094 #endif
00095 #if defined(__GNUC_MINOR__)
00096 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00097 #endif
00098 #if defined(__GNUC_PATCHLEVEL__)
00099 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00100 #endif
00101
00102 #elif defined(__PATHCC__)
00103 # define COMPILER_ID "PathScale"
00104 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00105 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00106 # if defined(__PATHCC_PATCHLEVEL__)
00107 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00108 # endif
00109
00110 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00111 # define COMPILER_ID "Embarcadero"
00112 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00113 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00114 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00115
00116 #elif defined(__BORLANDC__)
00117 # define COMPILER_ID "Borland"
00118   /* __BORLANDC__ = 0xVRR */
00119 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00120 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00121
00122 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00123 # define COMPILER_ID "Watcom"
00124    /* __WATCOMC__ = VVRR */
00125 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00126 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00127 # if (__WATCOMC__ % 10) > 0
00128 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00129 # endif
00130
00131 #elif defined(__WATCOMC__)
00132 # define COMPILER_ID "OpenWatcom"
00133    /* __WATCOMC__ = VVRP + 1100 */
00134 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00135 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00136 # if (__WATCOMC__ % 10) > 0
00137 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00138 # endif
00139
00140 #elif defined(__SUNPRO_CC)
00141 # define COMPILER_ID "SunPro"
00142 # if __SUNPRO_CC >= 0x5100
00143    /* __SUNPRO_CC = 0xVRRP */
00144 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»12)
00145 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xFF)
00146 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00147 # else
00148    /* __SUNPRO_CC = 0xVRP */
00149 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»8)
00150 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xF)
00151 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00152 # endif
00153
00154 #elif defined(__HP_aCC)
00155 # define COMPILER_ID "HP"
00156   /* __HP_aCC = VVRRPP */
00157 # define COMPILER_VERSION_MAJOR DEC(__HP_aCC/10000)
00158 # define COMPILER_VERSION_MINOR DEC(__HP_aCC/100 % 100)
00159 # define COMPILER_VERSION_PATCH DEC(__HP_aCC     % 100)
00160
00161 #elif defined(__DECCXX)
00162 # define COMPILER_ID "Compaq"
00163   /* __DECCXX_VER = VVRRTPPPP */
00164 # define COMPILER_VERSION_MAJOR DEC(__DECCXX_VER/10000000)
00165 # define COMPILER_VERSION_MINOR DEC(__DECCXX_VER/100000  % 100)
00166 # define COMPILER_VERSION_PATCH DEC(__DECCXX_VER         % 10000)
00167
00168 #elif defined(__IBMCPP__) && defined(__COMPILER_VER__)
00169 # define COMPILER_ID "zOS"
```

```
00170   /* __IBMCPP__ = VRP */
00171 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00172 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00173 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00174
00175 #elif defined(__open_xl__) && defined(__clang__)
00176 # define COMPILER_ID "IBMClang"
00177 # define COMPILER_VERSION_MAJOR DEC(__open_xl_version__)
00178 # define COMPILER_VERSION_MINOR DEC(__open_xl_release__)
00179 # define COMPILER_VERSION_PATCH DEC(__open_xl_modification__)
00180 # define COMPILER_VERSION_TWEAK DEC(__open_xl_ptf_fix_level__)
00181
00182
00183 #elif defined(__ibmxl__) && defined(__clang__)
00184 # define COMPILER_ID "XLClang"
00185 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00186 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00187 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00188 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00189
00190
00191 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ >= 800
00192 # define COMPILER_ID "XL"
00193   /* __IBMCPP__ = VRP */
00194 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00195 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00196 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00197
00198 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ < 800
00199 # define COMPILER_ID "VisualAge"
00200   /* __IBMCPP__ = VRP */
00201 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00202 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00203 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00204
00205 #elif defined(__NVCOMPILER)
00206 # define COMPILER_ID "NVHPC"
00207 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00208 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00209 # if defined(__NVCOMPILER_PATCHLEVEL__)
00210 #  define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00211 # endif
00212
00213 #elif defined(__PGI)
00214 # define COMPILER_ID "PGI"
00215 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00216 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00217 # if defined(__PGIC_PATCHLEVEL__)
00218 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00219 # endif
00220
00221 #elif defined(__clang__) && defined(__cray__)
00222 # define COMPILER_ID "CrayClang"
00223 # define COMPILER_VERSION_MAJOR DEC(__cray_major__)
00224 # define COMPILER_VERSION_MINOR DEC(__cray_minor__)
00225 # define COMPILER_VERSION_PATCH DEC(__cray_patchlevel__)
00226 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00227
00228
00229 #elif defined(_CRAYC)
00230 # define COMPILER_ID "Cray"
00231 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00232 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00233
00234 #elif defined(__TI_COMPILER_VERSION__)
00235 # define COMPILER_ID "TI"
00236   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00237 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00238 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00239 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00240
00241 #elif defined(__CLANG_FUJITSU)
00242 # define COMPILER_ID "FujitsuClang"
00243 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00244 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00245 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00246 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00247
00248
00249 #elif defined(__FUJITSU)
00250 # define COMPILER_ID "Fujitsu"
00251 # if defined(__FCC_version__)
00252 #   define COMPILER_VERSION __FCC_version__
00253 # elif defined(__FCC_major__)
00254 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00255 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00256 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
```

```
00257 # endif
00258 # if defined(__fcc_version)
00259 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00260 # elif defined(__FCC_VERSION)
00261 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00262 # endif
00263
00264
00265 #elif defined(__ghs__)
00266 # define COMPILER_ID "GHS"
00267 /* __GHS_VERSION_NUMBER = VVVVRP */
00268 # ifdef __GHS_VERSION_NUMBER
00269 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00270 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00271 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00272 # endif
00273
00274 #elif defined(__TASKING__)
00275 # define COMPILER_ID "Tasking"
00276   # define COMPILER_VERSION_MAJOR DEC(__VERSION__/1000)
00277   # define COMPILER_VERSION_MINOR DEC(__VERSION__ % 100)
00278 # define COMPILER_VERSION_INTERNAL DEC(__VERSION__)
00279
00280 #elif defined(__ORANGEC__)
00281 # define COMPILER_ID "OrangeC"
00282 # define COMPILER_VERSION_MAJOR DEC(__ORANGEC_MAJOR__)
00283 # define COMPILER_VERSION_MINOR DEC(__ORANGEC_MINOR__)
00284 # define COMPILER_VERSION_PATCH DEC(__ORANGEC_PATCHLEVEL__)
00285
00286 #elif defined(__SCO_VERSION__)
00287 # define COMPILER_ID "SCO"
00288
00289 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00290 # define COMPILER_ID "ARMCC"
00291 #if __ARMCC_VERSION >= 1000000
00292   /* __ARMCC_VERSION = VRRPPPP */
00293   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00294   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00295   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00296 #else
00297   /* __ARMCC_VERSION = VRPPPP */
00298   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00299   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00300   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00301 #endif
00302
00303
00304 #elif defined(__clang__) && defined(__apple_build_version__)
00305 # define COMPILER_ID "AppleClang"
00306 # if defined(_MSC_VER)
00307 #  define SIMULATE_ID "MSVC"
00308 # endif
00309 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00310 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00311 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00312 # if defined(_MSC_VER)
00313   /* _MSC_VER = VVRR */
00314 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00315 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00316 # endif
00317 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00318
00319 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00320 # define COMPILER_ID "ARMClang"
00321   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00322   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00323   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION/100   % 100)
00324 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00325
00326 #elif defined(__clang__) && defined(__ti__)
00327 # define COMPILER_ID "TIClang"
00328   # define COMPILER_VERSION_MAJOR DEC(__ti_major__)
00329   # define COMPILER_VERSION_MINOR DEC(__ti_minor__)
00330   # define COMPILER_VERSION_PATCH DEC(__ti_patchlevel__)
00331 # define COMPILER_VERSION_INTERNAL DEC(__ti_version__)
00332
00333 #elif defined(__clang__)
00334 # define COMPILER_ID "Clang"
00335 # if defined(_MSC_VER)
00336 #  define SIMULATE_ID "MSVC"
00337 # endif
00338 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00339 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00340 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00341 # if defined(_MSC_VER)
00342   /* _MSC_VER = VVRR */
00343 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
```

```
00344 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00345 # endif
00346
00347 #elif defined(__LCC__) && (defined(__GNUC__) || defined(__GNUG__) || defined(__MCST__))
00348 # define COMPILER_ID "LCC"
00349 # define COMPILER_VERSION_MAJOR DEC(__LCC__ / 100)
00350 # define COMPILER_VERSION_MINOR DEC(__LCC__ % 100)
00351 # if defined(__LCC_MINOR__)
00352 #  define COMPILER_VERSION_PATCH DEC(__LCC_MINOR__)
00353 # endif
00354 # if defined(__GNUC__) && defined(__GNUC_MINOR__)
00355 #  define SIMULATE_ID "GNU"
00356 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00357 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00358 #  if defined(__GNUC_PATCHLEVEL__)
00359 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00360 #  endif
00361 # endif
00362
00363 #elif defined(__GNUC__) || defined(__GNUG__)
00364 # define COMPILER_ID "GNU"
00365 # if defined(__GNUC__)
00366 #  define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00367 # else
00368 #  define COMPILER_VERSION_MAJOR DEC(__GNUG__)
00369 # endif
00370 # if defined(__GNUC_MINOR__)
00371 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00372 # endif
00373 # if defined(__GNUC_PATCHLEVEL__)
00374 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00375 # endif
00376
00377 #elif defined(_MSC_VER)
00378 # define COMPILER_ID "MSVC"
00379   /* _MSC_VER = VVRR */
00380 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00381 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00382 # if defined(_MSC_FULL_VER)
00383 #  if _MSC_VER >= 1400
00384     /* _MSC_FULL_VER = VVRRPPPPP */
00385 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00386 #  else
00387     /* _MSC_FULL_VER = VVRRPPPP */
00388 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00389 #  endif
00390 # endif
00391 # if defined(_MSC_BUILD)
00392 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00393 # endif
00394
00395 #elif defined(_ADI_COMPILER)
00396 # define COMPILER_ID "ADSP"
00397 #if defined(__VERSIONNUM__)
00398   /* __VERSIONNUM__ = 0xVVRRPTT */
00399 #  define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ >> 24 & 0xFF)
00400 #  define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ >> 16 & 0xFF)
00401 #  define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ >> 8 & 0xFF)
00402 #  define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00403 #endif
00404
00405 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00406 # define COMPILER_ID "IAR"
00407 # if defined(__VER__) && defined(__ICCARM__)
00408 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00409 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00410 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00411 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00412 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
      defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
      defined(__ICC8051__) || defined(__ICCSTM8__))
00413 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00414 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00415 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00416 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00417 # endif
00418
00419
00420 /* These compilers are either not known or too old to define an
00421   identification macro.  Try to identify the platform and guess that
00422   it is the native compiler.  */
00423 #elif defined(__hpux) || defined(__hpua)
00424 # define COMPILER_ID "HP"
00425
00426 #else /* unknown compiler */
00427 # define COMPILER_ID ""
00428 #endif
```

```
00429
00430 /* Construct the string literal in pieces to prevent the source from
00431    getting matched.  Store it in a pointer rather than an array
00432    because some compilers will just produce instructions to fill the
00433    array rather than assigning a pointer to a static array.  */
00434 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00435 #ifdef SIMULATE_ID
00436 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00437 #endif
00438
00439 #ifdef __QNXNTO__
00440 char const* qnxnto = "INFO" ":" "qnxnto[]";
00441 #endif
00442
00443 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00444 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00445 #endif
00446
00447 #define STRINGIFY_HELPER(X) #X
00448 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00449
00450 /* Identify known platforms by name.  */
00451 #if defined(__linux) || defined(__linux__) || defined(linux)
00452 # define PLATFORM_ID "Linux"
00453
00454 #elif defined(__MSYS__)
00455 # define PLATFORM_ID "MSYS"
00456
00457 #elif defined(__CYGWIN__)
00458 # define PLATFORM_ID "Cygwin"
00459
00460 #elif defined(__MINGW32__)
00461 # define PLATFORM_ID "MinGW"
00462
00463 #elif defined(__APPLE__)
00464 # define PLATFORM_ID "Darwin"
00465
00466 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00467 # define PLATFORM_ID "Windows"
00468
00469 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00470 # define PLATFORM_ID "FreeBSD"
00471
00472 #elif defined(__NetBSD__) || defined(__NetBSD)
00473 # define PLATFORM_ID "NetBSD"
00474
00475 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00476 # define PLATFORM_ID "OpenBSD"
00477
00478 #elif defined(__sun) || defined(sun)
00479 # define PLATFORM_ID "SunOS"
00480
00481 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00482 # define PLATFORM_ID "AIX"
00483
00484 #elif defined(__hpux) || defined(__hpux__)
00485 # define PLATFORM_ID "HP-UX"
00486
00487 #elif defined(__HAIKU__)
00488 # define PLATFORM_ID "Haiku"
00489
00490 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00491 # define PLATFORM_ID "BeOS"
00492
00493 #elif defined(__QNX__) || defined(__QNXNTO__)
00494 # define PLATFORM_ID "QNX"
00495
00496 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00497 # define PLATFORM_ID "Tru64"
00498
00499 #elif defined(__riscos) || defined(__riscos__)
00500 # define PLATFORM_ID "RISCos"
00501
00502 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00503 # define PLATFORM_ID "SINIX"
00504
00505 #elif defined(__UNIX_SV__)
00506 # define PLATFORM_ID "UNIX_SV"
00507
00508 #elif defined(__bsdos__)
00509 # define PLATFORM_ID "BSDOS"
00510
00511 #elif defined(_MPRAS) || defined(MPRAS)
00512 # define PLATFORM_ID "MP-RAS"
00513
00514 #elif defined(__osf) || defined(__osf__)
00515 # define PLATFORM_ID "OSF1"
```

```
00516
00517 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00518 # define PLATFORM_ID "SCO_SV"
00519
00520 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00521 # define PLATFORM_ID "ULTRIX"
00522
00523 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00524 # define PLATFORM_ID "Xenix"
00525
00526 #elif defined(__WATCOMC__)
00527 # if defined(__LINUX__)
00528 #  define PLATFORM_ID "Linux"
00529
00530 # elif defined(__DOS__)
00531 #  define PLATFORM_ID "DOS"
00532
00533 # elif defined(__OS2__)
00534 #  define PLATFORM_ID "OS2"
00535
00536 # elif defined(__WINDOWS__)
00537 #  define PLATFORM_ID "Windows3x"
00538
00539 # elif defined(__VXWORKS__)
00540 #  define PLATFORM_ID "VxWorks"
00541
00542 # else /* unknown platform */
00543 #  define PLATFORM_ID
00544 # endif
00545
00546 #elif defined(__INTEGRITY)
00547 # if defined(INT_178B)
00548 #  define PLATFORM_ID "Integrity178"
00549
00550 # else /* regular Integrity */
00551 #  define PLATFORM_ID "Integrity"
00552 # endif
00553
00554 # elif defined(_ADI_COMPILER)
00555 #  define PLATFORM_ID "ADSP"
00556
00557 #else /* unknown platform */
00558 # define PLATFORM_ID
00559
00560 #endif
00561
00562 /* For windows compilers MSVC and Intel we can determine
00563    the architecture of the compiler being used.  This is because
00564    the compilers do not have flags that can change the architecture,
00565    but rather depend on which compiler is being used
00566 */
00567 #if defined(_WIN32) && defined(_MSC_VER)
00568 # if defined(_M_IA64)
00569 #  define ARCHITECTURE_ID "IA64"
00570
00571 # elif defined(_M_ARM64EC)
00572 #  define ARCHITECTURE_ID "ARM64EC"
00573
00574 # elif defined(_M_X64) || defined(_M_AMD64)
00575 #  define ARCHITECTURE_ID "x64"
00576
00577 # elif defined(_M_IX86)
00578 #  define ARCHITECTURE_ID "X86"
00579
00580 # elif defined(_M_ARM64)
00581 #  define ARCHITECTURE_ID "ARM64"
00582
00583 # elif defined(_M_ARM)
00584 #  if _M_ARM == 4
00585 #   define ARCHITECTURE_ID "ARMV4I"
00586 #  elif _M_ARM == 5
00587 #   define ARCHITECTURE_ID "ARMV5I"
00588 #  else
00589 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00590 #  endif
00591
00592 # elif defined(_M_MIPS)
00593 #  define ARCHITECTURE_ID "MIPS"
00594
00595 # elif defined(_M_SH)
00596 #  define ARCHITECTURE_ID "SHx"
00597
00598 # else /* unknown architecture */
00599 #  define ARCHITECTURE_ID ""
00600 # endif
00601
00602 #elif defined(__WATCOMC__)
```

```
00603 # if defined(_M_I86)
00604 #  define ARCHITECTURE_ID "I86"
00605
00606 # elif defined(_M_IX86)
00607 #  define ARCHITECTURE_ID "X86"
00608
00609 # else /* unknown architecture */
00610 #  define ARCHITECTURE_ID ""
00611 # endif
00612
00613 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00614 # if defined(__ICCARM__)
00615 #  define ARCHITECTURE_ID "ARM"
00616
00617 # elif defined(__ICCRX__)
00618 #  define ARCHITECTURE_ID "RX"
00619
00620 # elif defined(__ICCRH850__)
00621 #  define ARCHITECTURE_ID "RH850"
00622
00623 # elif defined(__ICCRL78__)
00624 #  define ARCHITECTURE_ID "RL78"
00625
00626 # elif defined(__ICCRISCV__)
00627 #  define ARCHITECTURE_ID "RISCV"
00628
00629 # elif defined(__ICCAVR__)
00630 #  define ARCHITECTURE_ID "AVR"
00631
00632 # elif defined(__ICC430__)
00633 #  define ARCHITECTURE_ID "MSP430"
00634
00635 # elif defined(__ICCV850__)
00636 #  define ARCHITECTURE_ID "V850"
00637
00638 # elif defined(__ICC8051__)
00639 #  define ARCHITECTURE_ID "8051"
00640
00641 # elif defined(__ICCSTM8__)
00642 #  define ARCHITECTURE_ID "STM8"
00643
00644 # else /* unknown architecture */
00645 #  define ARCHITECTURE_ID ""
00646 # endif
00647
00648 #elif defined(__ghs__)
00649 # if defined(__PPC64__)
00650 #  define ARCHITECTURE_ID "PPC64"
00651
00652 # elif defined(__ppc__)
00653 #  define ARCHITECTURE_ID "PPC"
00654
00655 # elif defined(__ARM__)
00656 #  define ARCHITECTURE_ID "ARM"
00657
00658 # elif defined(__x86_64__)
00659 #  define ARCHITECTURE_ID "x64"
00660
00661 # elif defined(__i386__)
00662 #  define ARCHITECTURE_ID "X86"
00663
00664 # else /* unknown architecture */
00665 #  define ARCHITECTURE_ID ""
00666 # endif
00667
00668 #elif defined(__clang__) && defined(__ti__)
00669 # if defined(__ARM_ARCH)
00670 #  define ARCHITECTURE_ID "Arm"
00671
00672 # else /* unknown architecture */
00673 #  define ARCHITECTURE_ID ""
00674 # endif
00675
00676 #elif defined(__TI_COMPILER_VERSION__)
00677 # if defined(__TI_ARM__)
00678 #  define ARCHITECTURE_ID "ARM"
00679
00680 # elif defined(__MSP430__)
00681 #  define ARCHITECTURE_ID "MSP430"
00682
00683 # elif defined(__TMS320C28XX__)
00684 #  define ARCHITECTURE_ID "TMS320C28x"
00685
00686 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00687 #  define ARCHITECTURE_ID "TMS320C6x"
00688
00689 # else /* unknown architecture */
```

```
00690 #  define ARCHITECTURE_ID ""
00691 # endif
00692
00693 # elif defined(__ADSPSHARC__)
00694 #  define ARCHITECTURE_ID "SHARC"
00695
00696 # elif defined(__ADSPBLACKFIN__)
00697 #  define ARCHITECTURE_ID "Blackfin"
00698
00699 #elif defined(__TASKING__)
00700
00701 # if defined(__CTC__) || defined(__CPTC__)
00702 #  define ARCHITECTURE_ID "TriCore"
00703
00704 # elif defined(__CMCS__)
00705 #  define ARCHITECTURE_ID "MCS"
00706
00707 # elif defined(__CARM__)
00708 #  define ARCHITECTURE_ID "ARM"
00709
00710 # elif defined(__CARC__)
00711 #  define ARCHITECTURE_ID "ARC"
00712
00713 # elif defined(__C51__)
00714 #  define ARCHITECTURE_ID "8051"
00715
00716 # elif defined(__CPCP__)
00717 #  define ARCHITECTURE_ID "PCP"
00718
00719 # else
00720 #  define ARCHITECTURE_ID ""
00721 # endif
00722
00723 #else
00724 #  define ARCHITECTURE_ID
00725 #endif
00726
00727 /* Convert integer to decimal digit literals.  */
00728 #define DEC(n)                   \
00729   ('0' + (((n) / 10000000)%10)), \
00730   ('0' + (((n) / 1000000)%10)),  \
00731   ('0' + (((n) / 100000)%10)),   \
00732   ('0' + (((n) / 10000)%10)),    \
00733   ('0' + (((n) / 1000)%10)),     \
00734   ('0' + (((n) / 100)%10)),      \
00735   ('0' + (((n) / 10)%10)),       \
00736   ('0' +  ((n) % 10))
00737
00738 /* Convert integer to hex digit literals.  */
00739 #define HEX(n)             \
00740   ('0' + ((n)>>28 & 0xF)), \
00741   ('0' + ((n)>>24 & 0xF)), \
00742   ('0' + ((n)>>20 & 0xF)), \
00743   ('0' + ((n)>>16 & 0xF)), \
00744   ('0' + ((n)>>12 & 0xF)), \
00745   ('0' + ((n)>>8  & 0xF)), \
00746   ('0' + ((n)>>4  & 0xF)), \
00747   ('0' + ((n)     & 0xF))
00748
00749 /* Construct a string literal encoding the version number. */
00750 #ifdef COMPILER_VERSION
00751 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "]";
00752
00753 /* Construct a string literal encoding the version number components. */
00754 #elif defined(COMPILER_VERSION_MAJOR)
00755 char const info_version[] = {
00756   'I', 'N', 'F', 'O', ':',
00757   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00758   COMPILER_VERSION_MAJOR,
00759 # ifdef COMPILER_VERSION_MINOR
00760   '.', COMPILER_VERSION_MINOR,
00761 #  ifdef COMPILER_VERSION_PATCH
00762   '.', COMPILER_VERSION_PATCH,
00763 #   ifdef COMPILER_VERSION_TWEAK
00764     '.', COMPILER_VERSION_TWEAK,
00765 #   endif
00766 #  endif
00767 # endif
00768   ']','\0'};
00769 #endif
00770
00771 /* Construct a string literal encoding the internal version number. */
00772 #ifdef COMPILER_VERSION_INTERNAL
00773 char const info_version_internal[] = {
00774   'I', 'N', 'F', 'O', ':',
00775   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00776   'i','n','t','e','r','n','a','l','[',
```

```
00777   COMPILER_VERSION_INTERNAL,']','\0'};
00778 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00779 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
      COMPILER_VERSION_INTERNAL_STR "]";
00780 #endif
00781
00782 /* Construct a string literal encoding the version number components. */
00783 #ifdef SIMULATE_VERSION_MAJOR
00784 char const info_simulate_version[] = {
00785   'I', 'N', 'F', 'O', ':',
00786   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00787   SIMULATE_VERSION_MAJOR,
00788 # ifdef SIMULATE_VERSION_MINOR
00789   '.', SIMULATE_VERSION_MINOR,
00790 #  ifdef SIMULATE_VERSION_PATCH
00791    '.', SIMULATE_VERSION_PATCH,
00792 #   ifdef SIMULATE_VERSION_TWEAK
00793     '.', SIMULATE_VERSION_TWEAK,
00794 #   endif
00795 #  endif
00796 # endif
00797   ']','\0'};
00798 #endif
00799
00800 /* Construct the string literal in pieces to prevent the source from
00801    getting matched.  Store it in a pointer rather than an array
00802    because some compilers will just produce instructions to fill the
00803    array rather than assigning a pointer to a static array.  */
00804 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00805 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00806
00807
00808
00809 #define CXX_STD_98 199711L
00810 #define CXX_STD_11 201103L
00811 #define CXX_STD_14 201402L
00812 #define CXX_STD_17 201703L
00813 #define CXX_STD_20 202002L
00814 #define CXX_STD_23 202302L
00815
00816 #if defined(__INTEL_COMPILER) && defined(_MSVC_LANG)
00817 #  if _MSVC_LANG > CXX_STD_17
00818 #    define CXX_STD _MSVC_LANG
00819 #  elif _MSVC_LANG == CXX_STD_17 && defined(__cpp_aggregate_paren_init)
00820 #    define CXX_STD CXX_STD_20
00821 #  elif _MSVC_LANG > CXX_STD_14 && __cplusplus > CXX_STD_17
00822 #    define CXX_STD CXX_STD_20
00823 #  elif _MSVC_LANG > CXX_STD_14
00824 #    define CXX_STD CXX_STD_17
00825 #  elif defined(__INTEL_CXX11_MODE__) && defined(__cpp_aggregate_nsdmi)
00826 #    define CXX_STD CXX_STD_14
00827 #  elif defined(__INTEL_CXX11_MODE__)
00828 #    define CXX_STD CXX_STD_11
00829 #  else
00830 #    define CXX_STD CXX_STD_98
00831 #  endif
00832 #elif defined(_MSC_VER) && defined(_MSVC_LANG)
00833 #  if _MSVC_LANG > __cplusplus
00834 #    define CXX_STD _MSVC_LANG
00835 #  else
00836 #    define CXX_STD __cplusplus
00837 #  endif
00838 #elif defined(__NVCOMPILER)
00839 #  if __cplusplus == CXX_STD_17 && defined(__cpp_aggregate_paren_init)
00840 #    define CXX_STD CXX_STD_20
00841 #  else
00842 #    define CXX_STD __cplusplus
00843 #  endif
00844 #elif defined(__INTEL_COMPILER) || defined(__PGI)
00845 #  if __cplusplus == CXX_STD_11 && defined(__cpp_namespace_attributes)
00846 #    define CXX_STD CXX_STD_17
00847 #  elif __cplusplus == CXX_STD_11 && defined(__cpp_aggregate_nsdmi)
00848 #    define CXX_STD CXX_STD_14
00849 #  else
00850 #    define CXX_STD __cplusplus
00851 #  endif
00852 #elif (defined(__IBMCPP__) || defined(__ibmxl__)) && defined(__linux__)
00853 #  if __cplusplus == CXX_STD_11 && defined(__cpp_aggregate_nsdmi)
00854 #    define CXX_STD CXX_STD_14
00855 #  else
00856 #    define CXX_STD __cplusplus
00857 #  endif
00858 #elif __cplusplus == 1 && defined(__GXX_EXPERIMENTAL_CXX0X__)
00859 #  define CXX_STD CXX_STD_11
00860 #else
00861 #  define CXX_STD __cplusplus
00862 #endif
```

```
00863
00864 const char* info_language_standard_default = "INFO" ":" "standard_default["
00865 #if CXX_STD > CXX_STD_23
00866   "26"
00867 #elif CXX_STD > CXX_STD_20
00868   "23"
00869 #elif CXX_STD > CXX_STD_17
00870   "20"
00871 #elif CXX_STD > CXX_STD_14
00872   "17"
00873 #elif CXX_STD > CXX_STD_11
00874   "14"
00875 #elif CXX_STD >= CXX_STD_11
00876   "11"
00877 #else
00878   "98"
00879 #endif
00880 "]";
00881
00882 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00883 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlC__) ||           \
00884      defined(__TI_COMPILER_VERSION__)) &&                                     \
00885   !defined(__STRICT_ANSI__)
00886   "ON"
00887 #else
00888   "OFF"
00889 #endif
00890 "]";
00891
00892 /*--------------------------------------------------------------------------*/
00893
00894 int main(int argc, char* argv[])
00895 {
00896   int require = 0;
00897   require += info_compiler[argc];
00898   require += info_platform[argc];
00899   require += info_arch[argc];
00900 #ifdef COMPILER_VERSION_MAJOR
00901   require += info_version[argc];
00902 #endif
00903 #ifdef COMPILER_VERSION_INTERNAL
00904   require += info_version_internal[argc];
00905 #endif
00906 #ifdef SIMULATE_ID
00907   require += info_simulate[argc];
00908 #endif
00909 #ifdef SIMULATE_VERSION_MAJOR
00910   require += info_simulate_version[argc];
00911 #endif
00912 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00913   require += info_cray[argc];
00914 #endif
00915   require += info_language_standard_default[argc];
00916   require += info_language_extensions_default[argc];
00917   (void)argv;
00918   return require;
00919 }
```

6.9 build/CMakeFiles/RPN.dir/main.cpp.obj.d File Reference

6.10 main.cpp.obj.d

Go to the documentation of this file.

```
00001 CMakeFiles/RPN.dir/main.cpp.obj: C:\Users\Blixon\Desktop\RPN\main.cpp \
00002  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\iostream \
00003  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\c++config.h \
00004  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\os_defines.h \
00005  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\cpu_defines.h \
00006  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ostream \
00007  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ios \
00008  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\iosfwd \
00009  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stringfwd.h \
00010  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\memoryfwd.h \
00011  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\postypes.h \
00012  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cwchar \
00013  c:\mingw\include\wchar.h c:\mingw\include\_mingw.h \
00014  c:\mingw\include\msvcrtver.h c:\mingw\include\w32api.h \
00015  c:\mingw\include\sdkddkver.h c:\mingw\include\features.h \
```

```
00016  c:\mingw\include\wctype.h \
00017  c:\mingw\lib\gcc\mingw32\9.2.0\include\stddef.h \
00018  c:\mingw\include\sys\types.h c:\mingw\include\stdio.h \
00019  c:\mingw\include\sys/types.h \
00020  c:\mingw\lib\gcc\mingw32\9.2.0\include\stdarg.h \
00021  c:\mingw\include\stdlib.h c:\mingw\include\direct.h \
00022  c:\mingw\include\sys/stat.h c:\mingw\include\conio.h \
00023  c:\mingw\include\io.h c:\mingw\include\stdint.h c:\mingw\include\time.h \
00024  c:\mingw\include\locale.h c:\mingw\include\process.h \
00025  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\exception \
00026  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\exception.h \
00027  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\exception_ptr.h \
00028  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\exception_defines.h \
00029  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\cxxabi_init_exception.h \
00030  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\typeinfo \
00031  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\hash_bytes.h \
00032  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\new \
00033  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\nested_exception.h \
00034  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\move.h \
00035  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\concept_check.h \
00036  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\type_traits \
00037  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\char_traits.h \
00038  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_algobase.h \
00039  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\functexcept.h \
00040  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\cpp_type_traits.h \
00041  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\type_traits.h \
00042  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\numeric_traits.h \
00043  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_pair.h \
00044  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_iterator_base_types.h \
00045  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_iterator_base_funcs.h \
00046  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\debug\assertions.h \
00047  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_iterator.h \
00048  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ptr_traits.h \
00049  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\debug\debug.h \
00050  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\predefined_ops.h \
00051  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cstdint \
00052  c:\mingw\lib\gcc\mingw32\9.2.0\include\stdint.h \
00053  c:\mingw\include\stdint.h \
00054  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\localefwd.h \
00055  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\c++locale.h \
00056  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\clocale \
00057  c:\mingw\include\locale.h \
00058  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cctype \
00059  c:\mingw\include\ctype.h c:\mingw\include\wctype.h \
00060  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ios_base.h \
00061  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\atomicity.h \
00062  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\gthr.h \
00063  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\gthr-default.h \
00064  c:\mingw\include\errno.h \
00065  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\atomic_word.h \
00066  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_classes.h \
00067  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\string \
00068  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\allocator.h \
00069  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\c++allocator.h \
00070  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\new_allocator.h \
00071  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ostream_insert.h \
00072  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\cxxabi_forced.h \
00073  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_function.h \
00074  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\backward\binders.h \
00075  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\range_access.h \
00076  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\initializer_list \
00077  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_string.h \
00078  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\alloc_traits.h \
00079  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\alloc_traits.h \
00080  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\string_conversions.h \
00081  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cstdlib \
00082  c:\mingw\include\stdlib.h c:\mingw\include\errno.h \
00083  c:\mingw\include\alloca.h \
00084  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\std_abs.h \
00085  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cstdio \
00086  c:\mingw\include\stdio.h \
00087  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cerrno \
00088  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\functional_hash.h \
00089  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_string.tcc \
00090  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_classes.tcc \
00091  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\system_error \
00092  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\error_constants.h \
00093  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\stdexcept \
00094  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\streambuf \
00095  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\streambuf.tcc \
00096  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_ios.h \
00097  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_facets.h \
00098  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cwctype \
00099  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\ctype_base.h \
00100  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\streambuf_iterator.h \
00101  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\ctype_inline.h \
00102  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_facets.tcc \
```

```
00103  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_ios.tcc \
00104  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ostream.tcc \
00105  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\istream \
00106  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\istream.tcc \
00107  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\fstream \
00108  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\codecvt.h \
00109  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\basic_file.h \
00110  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\c++io.h \
00111  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\fstream.tcc \
00112  C:/Users/Blixon/Desktop/RPN/lib/RPN.h \
00113  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\unordered_set \
00114  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\aligned_buffer.h \
00115  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\hashtable.h \
00116  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\hashtable_policy.h \
00117  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\tuple \
00118  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\utility \
00119  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_relops.h \
00120  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\array \
00121  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\uses_allocator.h \
00122  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\invoke.h \
00123  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\limits \
00124  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\unordered_set.h \
00125  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\erase_if.h \
00126  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\sstream \
00127  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\sstream.tcc
```

6.11 build/lib/CMakeFiles/RPN_LIB.dir/RPN.cpp.obj.d File Reference

6.12 RPN.cpp.obj.d

Go to the documentation of this file.

```
00001 lib/CMakeFiles/RPN_LIB.dir/RPN.cpp.obj: \
00002  C:\Users\Blixon\Desktop\RPN\lib\RPN.cpp \
00003  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\iostream \
00004  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\c++config.h \
00005  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\os_defines.h \
00006  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\cpu_defines.h \
00007  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ostream \
00008  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ios \
00009  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\iosfwd \
00010  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stringfwd.h \
00011  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\memoryfwd.h \
00012  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\postypes.h \
00013  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cwchar \
00014  c:\mingw\include\wchar.h c:\mingw\include\_mingw.h \
00015  c:\mingw\include\msvcrtver.h c:\mingw\include\w32api.h \
00016  c:\mingw\include\sdkddkver.h c:\mingw\include\features.h \
00017  c:\mingw\include\wctype.h \
00018  c:\mingw\lib\gcc\mingw32\9.2.0\include\stddef.h \
00019  c:\mingw\include\sys\types.h c:\mingw\include\stdio.h \
00020  c:\mingw\include\sys/types.h \
00021  c:\mingw\lib\gcc\mingw32\9.2.0\include\stdarg.h \
00022  c:\mingw\include\stdlib.h c:\mingw\include\direct.h \
00023  c:\mingw\include\sys/stat.h c:\mingw\include\conio.h \
00024  c:\mingw\include\io.h c:\mingw\include\stdint.h c:\mingw\include\time.h \
00025  c:\mingw\include\locale.h c:\mingw\include\process.h \
00026  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\exception \
00027  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\exception.h \
00028  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\exception_ptr.h \
00029  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\exception_defines.h \
00030  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\cxxabi_init_exception.h \
00031  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\typeinfo \
00032  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\hash_bytes.h \
00033  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\new \
00034  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\nested_exception.h \
00035  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\move.h \
00036  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\concept_check.h \
00037  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\type_traits \
00038  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\char_traits.h \
00039  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_algobase.h \
00040  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\functexcept.h \
00041  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\cpp_type_traits.h \
00042  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\type_traits.h \
00043  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\numeric_traits.h \
00044  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_pair.h \
00045  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_iterator_base_types.h \
00046  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_iterator_base_funcs.h \
00047  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\debug\assertions.h \
```

```
00048  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_iterator.h \
00049  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ptr_traits.h \
00050  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\debug\debug.h \
00051  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\predefined_ops.h \
00052  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cstdint \
00053  c:\mingw\lib\gcc\mingw32\9.2.0\include\stdint.h \
00054  c:\mingw\include\stdint.h \
00055  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\localefwd.h \
00056  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\c++locale.h \
00057  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\clocale \
00058  c:\mingw\include\locale.h \
00059  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cctype \
00060  c:\mingw\include\ctype.h c:\mingw\include\wctype.h \
00061  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ios_base.h \
00062  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\atomicity.h \
00063  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\gthr.h \
00064  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\gthr-default.h \
00065  c:\mingw\include\errno.h \
00066  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\atomic_word.h \
00067  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_classes.h \
00068  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\string \
00069  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\allocator.h \
00070  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\c++allocator.h \
00071  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\new_allocator.h \
00072  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ostream_insert.h \
00073  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\cxxabi_forced.h \
00074  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_function.h \
00075  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\backward\binders.h \
00076  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\range_access.h \
00077  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\initializer_list \
00078  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_string.h \
00079  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\alloc_traits.h \
00080  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\alloc_traits.h \
00081  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\string_conversions.h \
00082  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cstdlib \
00083  c:\mingw\include\stdlib.h c:\mingw\include\errno.h \
00084  c:\mingw\include\alloca.h \
00085  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\std_abs.h \
00086  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cstdio \
00087  c:\mingw\include\stdio.h \
00088  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cerrno \
00089  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\functional_hash.h \
00090  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_string.tcc \
00091  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_classes.tcc \
00092  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\system_error \
00093  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\error_constants.h \
00094  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\stdexcept \
00095  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\streambuf \
00096  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\streambuf.tcc \
00097  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_ios.h \
00098  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_facets.h \
00099  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cwctype \
00100  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\ctype_base.h \
00101  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\streambuf_iterator.h \
00102  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\ctype_inline.h \
00103  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_facets.tcc \
00104  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\basic_ios.tcc \
00105  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\ostream.tcc \
00106  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\istream \
00107  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\istream.tcc \
00108  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\memory \
00109  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_construct.h \
00110  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_uninitialized.h \
00111  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_tempbuf.h \
00112  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_raw_storage_iter.h \
00113  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\concurrence.h \
00114  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\uses_allocator.h \
00115  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\unique_ptr.h \
00116  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\utility \
00117  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_relops.h \
00118  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\tuple \
00119  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\array \
00120  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\invoke.h \
00121  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\shared_ptr.h \
00122  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\shared_ptr_base.h \
00123  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\allocated_ptr.h \
00124  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\refwrap.h \
00125  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ext\aligned_buffer.h \
00126  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\shared_ptr_atomic.h \
00127  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\atomic_base.h \
00128  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\atomic_lockfree_defines.h \
00129  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\backward\auto_ptr.h \
00130  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\sstream \
00131  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\sstream.tcc \
00132  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\stack \
00133  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\deque \
00134  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_deque.h \
```

```
00135  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\deque.tcc \
00136  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_stack.h \
00137  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\unordered_set \
00138  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\hashtable.h \
00139  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\hashtable_policy.h \
00140  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\limits \
00141  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\unordered_set.h \
00142  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\erase_if.h \
00143  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\climits \
00144  c:\mingw\include\limits.h \
00145  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cmath c:\mingw\include\math.h \
00146  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\map \
00147  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_tree.h \
00148  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_map.h \
00149  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_multimap.h \
00150  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\regex \
00151  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\algorithm \
00152  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_algo.h \
00153  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\algorithmfwd.h \
00154  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_heap.h \
00155  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\uniform_int_dist.h \
00156  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bitset \
00157  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\iterator \
00158  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stream_iterator.h \
00159  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\locale \
00160  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_facets_nonio.h \
00161  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\ctime c:\mingw\include\time.h \
00162  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\time_members.h \
00163  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\mingw32\bits\messages_members.h \
00164  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\codecvt.h \
00165  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_facets_nonio.tcc \
00166  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\locale_conv.h \
00167  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\vector \
00168  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_vector.h \
00169  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\stl_bvector.h \
00170  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\vector.tcc \
00171  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\cstring \
00172  c:\mingw\include\string.h c:\mingw\include\strings.h \
00173  c:\mingw\include\wchar.h \
00174  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\std_function.h \
00175  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_constants.h \
00176  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_error.h \
00177  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_automaton.h \
00178  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_automaton.tcc \
00179  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_scanner.h \
00180  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_scanner.tcc \
00181  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_compiler.h \
00182  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_compiler.tcc \
00183  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex.h \
00184  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex.tcc \
00185  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_executor.h \
00186  c:\mingw\lib\gcc\mingw32\9.2.0\include\c++\bits\regex_executor.tcc \
00187  C:\Users\Blixon\Desktop\RPN\lib\RPN.h
```

6.13 lib/RPN.cpp File Reference

```
#include <iostream>
#include <memory>
#include <sstream>
#include <string>
#include <stack>
#include <unordered_set>
#include <climits>
#include <cmath>
#include <map>
#include <regex>
#include "RPN.h"
```

Include dependency graph for RPN.cpp:

6.14 RPN.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <memory>
00003 #include <sstream>
00004 #include <string>
00005 #include <stack>
00006 #include <unordered_set>
00007 #include <climits>
00008 #include <cmath>
00009 #include <map>
00010 #include <regex>
00011 #include "RPN.h"
00012
00013 namespace RPN {
00021     int sumLetters(const std::string& str) {
00022         int sum=0;
00023         for (const auto letter : str) {
00024             sum+=letter;
00025         }
00026         return sum;
00027     }
00028
00035     double handleDivision(const double& a, const double& b) {
00036         if (b == 0) {
00037             std::cerr << "[ " << a << "/" << b << " ] - illegal division (Division by zero).\n";
00038             std::exit(1);
00039         }
00040         return a/b;
00041     }
00042
00048     double handleSqrt(const double& a) {
00049         if (a < 0) {
00050             std::cerr << "[ sqrt(" << a << ") ] - unsupported root (Root of negative number).\n";
00051             std::exit(1);
00052         }
00053         return std::sqrt(a);
00054     }
00055
00061     double handleCbrt(const double& a) {
00062         if (a < 0) {
00063             std::cerr << "[ cbrt(" << a << ") ] - unsupported root (Root of negative number).\n";
00064             std::exit(1);
00065         }
00066         return std::cbrt(a);
00067     }
00068
00076     double calculate(const double& a, const double& b, const std::string& op) {
00077         switch (op[0]) {
00078             case '*':
00079                 return a*b;
00080             case '/':
00081                 return handleDivision(a, b);
00082             case '\\':
00083                 return handleDivision(b, a);
00084             case '+':
00085                 return a+b;
00086             case '-':
00087                 return a-b;
00088             case '^':
00089                 return std::pow(a, b);
00090             default:
00091                 return LONG_MIN;
00092         }
00093     }
00094
00101     double calculate(const double& a, const std::string& op) {
00106         switch (sumLetters(op)) {
00107             case 458: //sqrt
00108                 return handleSqrt(a);
00109             case 427: //cbrt
00110                 return handleCbrt(a);
00111             case 310: //abs
00112                 return std::abs(a);
00113             case 330: //sin
00114                 return std::sin(a);
00115             case 325: //cos
00116                 return std::cos(a);
00117             case 323: //tan
00118                 return std::tan(a);
00119             default:
00120                 return LONG_MIN;
00121         }
00122     }
00123
00127     constexpr int EXP_PREC = 100;
00131     constexpr int TRIG_FUN_PREC = EXP_PREC-1;
00135     constexpr int MULT_DIV_PREC = TRIG_FUN_PREC-1;
00139     constexpr int ADD_SUB_PREC = MULT_DIV_PREC-1;
```

```
00140
00144     const std::map<std::string, int> operatorPrecedence = {
00145         {"^", EXP_PREC},
00146         {"sqrt", EXP_PREC},
00147         {"cbrt", EXP_PREC},
00148         {"sin", TRIG_FUN_PREC},
00149         {"cos", TRIG_FUN_PREC},
00150         {"tan", TRIG_FUN_PREC},
00151         {"*", MULT_DIV_PREC},
00152         {"/", MULT_DIV_PREC},
00153         {"\\", MULT_DIV_PREC},
00154         {"+", ADD_SUB_PREC},
00155         {"-", ADD_SUB_PREC},
00156     };
00157
00161     const std::unordered_set<std::string> one_arg_operators = {
00162         "sqrt",
00163         "cbrt",
00164         "sin",
00165         "cos",
00166         "tan",
00167     };
00168
00172     const std::unordered_set<std::string> two_arg_operators = {
00173         "^",
00174         "*",
00175         "/",
00176         "\\",
00177         "+",
00178         "-"
00179     };
00180
00186     bool isOperator(const std::string& op) {
00187         return operatorPrecedence.count(op) > 0;
00188     }
00189
00195     bool is1ArgOperator(const std::string& op) {
00196         return one_arg_operators.count(op) > 0;
00197     }
00198
00204     bool is2ArgOperator(const std::string& op) {
00205         return two_arg_operators.count(op) > 0;
00206     }
00207
00208     double RPNSolver::getResult(const std::string& equation) {
00209         TokenReader reader(equation);
00210
00211         std::string token;
00212         std::stack<double> numbers;
00213         while (!(token = reader.next()).empty()) {
00214             if (is1ArgOperator(token)) {
00215                 double& a = numbers.top();
00216                 a = calculate(a, token);
00217             } else if (is2ArgOperator(token)) {
00224                 double b = numbers.top();
00225                 numbers.pop();
00226                 double& a = numbers.top();
00227                 a = calculate(a, b, token);
00228             } else {
00229                 numbers.push(std::stod(token));
00230             }
00231         }
00237         return  numbers.top();
00238     }
00239
00240     TokenReader::TokenReader(const std::string& string) {
00241         string_ = string;
00242         stream = std::stringstream(string);
00243     }
00244
00245     std::string TokenReader::next() {
00246         std::string token;
00247         stream » token;
00248         return token;
00249     }
00250
00251     std::string TokenReader::getString() {
00252         return string_;
00253     }
00254
00255     bool TokenReader::finished() const {
00256         return stream.eof();
00257     }
00258
00259     std::string TokenReader::peek() {
00260         std::streampos currentPos = stream.tellg();
00261         std::string next = this->next();
```

```
00262         stream.seekg(currentPos);
00263         return next;
00264     }
00265
00266     std::string NotationConverter::aopb(const std::string &a, const std::string &b, const std::string
      &op) {
00267         std::string combined = a;
00268         combined.append(" ");
00269         combined.append(op);
00270         combined.append(" ");
00271         combined.append(b);
00272         return combined;
00273     }
00274
00275     std::string NotationConverter::wrapInParentheses(const std::string &a, const std::string &b, const
      std::string &op) {
00276         std::string combined = "( ";
00277         std::string aopbStr = aopb(a, b, op);
00278         combined.append(aopbStr);
00279         combined.append(" )");
00280         return combined;
00281     }
00282
00283     std::string NotationConverter::onlyParentheses(const std::string &a) {
00284         std::string combined = "( ";
00285         combined.append(a);
00286         combined.append(" )");
00287         return combined;
00288     }
00289
00290     std::string NotationConverter::infixToRPN(const std::string &infix) {
00291         std::string equation;
00292         std::stack<std::string> operators;
00293         TokenReader reader(infix);
00294
00295         while (!reader.finished()) {
00296             std::string token = reader.next();
00297             if (isOperator(token)) {
00298                 if (!operators.empty() && operators.top() != "(") { // If stack not empty and newest
      is not (
00299                     std::string onStack = operators.top();
00300                     if (operatorPrecedence.at(onStack) >= operatorPrecedence.at(token)) {
00301                         equation.append(onStack);
00302                         equation.append(" ");
00303                         operators.pop();
00304                     }
00305                 }
00306                 operators.push(token);
00307             } else if (token == "(") {
00308                 operators.push(token);
00309             } else if (token == ")") {
00310                 while (operators.top() != "(") {
00311                     std::string op = operators.top();
00312                     equation.append(op);
00313                     equation.append(" ");
00314                     operators.pop();
00315                 }
00316                 operators.pop();
00317             } else {
00318                 equation.append(token);
00319                 equation.append(" ");
00320             }
00321         }
00322         while (!operators.empty()) {
00323             std::string op = operators.top();
00324             equation.append(op);
00325             equation.append(" ");
00326             operators.pop();
00327         }
00328         return equation;
00329     }
00330
00331     std::string NotationConverter::RPNtoInfix(const std::string &RPN) {
00332         TokenReader reader(RPN);
00333         std::stack<std::string> infixStack;
00334         while (!reader.finished()) {
00335             std::string token = reader.next();
00336             if (is1ArgOperator(token)) {
00337                 const std::string& operand = infixStack.top();
00338                 std::string inOperator = token;
00339                 inOperator.append(" ");
00340                 inOperator.append(onlyParentheses(operand));
00341                 infixStack.top() = inOperator;
00342             } else if (is2ArgOperator(token)) {
00343                 std::string rightOperand = infixStack.top();
00344                 infixStack.pop();
00345                 std::string& leftOperand = infixStack.top();
```

```
00346
00347                 bool needsParentheses = false;
00348                 if (!reader.finished()) {
00349                     std::string nextToken = reader.peek();
00350                     if (!isOperator(nextToken)) {
00351                         needsParentheses = true;
00352                     } else {
00353                         needsParentheses = operatorPrecedence.at(token) <
    operatorPrecedence.at(nextToken);
00354                     }
00355                 }
00356
00357                 leftOperand = needsParentheses
00358                             ? wrapInParentheses(leftOperand, rightOperand, token)
00359                             : aopb(leftOperand, rightOperand, token);
00360             } else {
00361                 infixStack.push(token);
00362             }
00363         }
00364         return infixStack.top();
00365     }
00366
00367     bool NotationDeterminer::isRPN(const std::string &equation) {
00368         TokenReader reader(equation);
00369         std::string lastToken;
00370         while (!reader.finished()) {
00371             lastToken = reader.next();
00372         }
00373         // Last token in RPN is always an operator.
00374         return isOperator(lastToken);
00375     }
00376
00377     bool NotationDeterminer::isInfix(const std::string &equation) {
00378         return !isRPN(equation);
00379     }
00380
00381     std::string Spacer::addSpacesAroundParentheses(const std::string& input) {
00382         std::string result;
00383         for (const char ch : input) {
00384             if (ch == '(' || ch == ')') {
00385                 result += ' ';
00386                 result += ch;
00387                 result += ' ';
00388             } else {
00389                 result += ch;
00390             }
00391         }
00392         return result;
00393     }
00394
00395     std::string Spacer::removeSpacesAroundParentheses(const std::string &input) {
00396         std::string result = input;
00397
00398         // Remove all spaces between '('
00399         result = std::regex_replace(result, std::regex(R"(\s*\(\s*)"), "(");
00400
00401         // Remove all spaces between ')'
00402         result = std::regex_replace(result, std::regex(R"(\s*\)\s*)"), ")");
00403
00404         return result;
00405     }
00406
00407     std::string Spacer::addSpacesAroundOperators(const std::string &input) {
00408         std::string result = input;
00409
00410         for (const auto& entry : operatorPrecedence) {
00411             const std::string& key = entry.first;
00412             std::string pattern;
00413             // Escape math operators that are regex tokens
00414             if (key == "+" || key == "*" || key == "\\" || key == "^") {
00415                 pattern = "\\" + key;
00416             } else {
00417                 pattern = key;
00418             }
00419             std::string replacement = " " + key + " ";
00420             result = std::regex_replace(result, std::regex(pattern), replacement);
00421         }
00422
00423         return result;
00424     }
00425
00426     std::string Spacer::removeSpacesAroundOperators(const std::string& input) {
00427         std::string result = input;
00428
00429         for (const auto& entry : operatorPrecedence) {
00430             const std::string& key = entry.first;
00431             std::string pattern;
```

```
00432
00433             // Escape math operators that are regex tokens
00434             if (key == "+" || key == "*" || key == "\\" || key == "^") {
00435                 pattern = "\\s*\\" + key + "\\s*";
00436             } else {
00437                 pattern = "\\s*" + key + "\\s*";
00438             }
00439
00440             result = std::regex_replace(result, std::regex(pattern), key);
00441         }
00442
00443         return result;
00444     }
00445
00446     std::string Spacer::mergeSpaces(const std::string &input) {
00447         return std::regex_replace(input, std::regex("\\s+"), " ");
00448     }
00449
00450
00451     bool EquationValidator::is_number(const std::string &str) {
00452         std::istringstream iss(str);
00453         double d;
00454         return iss >> std::noskipws >> d && iss.eof();
00455     }
00456
00457     bool EquationValidator::isValidRPN(const std::string &equation) {
00458         std::stack<int> operandStack;
00459         TokenReader reader(equation);
00460
00461         while (!reader.finished()) {
00462             std::string token = reader.next();
00463             if (token.empty()) break; //End of equation
00464
00465             if (is1ArgOperator(token)) {
00466                 if (operandStack.size() < 1) return false;
00467                 // Consumes operand and returns to the stack
00468                 // 1 element. Therefore, stack count stays the same.
00469             }
00470             else if (is2ArgOperator(token)) {
00471                 if (operandStack.size() < 2) return false;
00472                 operandStack.pop();
00473                 // Consumes 1 more element and returns 1 element back.
00474                 // Therefore, 1 pop is sufficient.
00475             }
00476             else {
00477                 try {
00478                     std::stod(token);
00479                     operandStack.push(1);
00480                 }
00481                 catch (...) {
00482                     // Invalid token
00483                     return false;
00484                 }
00485             }
00486         }
00487
00488         return operandStack.size() == 1;
00489     }
00490
00491     bool EquationValidator::isValidInfix(const std::string& equation) {
00492         TokenReader reader(equation);
00493         std::stack<std::string> parentheses;
00494         int operandCount = 0;
00495         int operatorCount = 0;
00496
00497         while (!reader.finished()) {
00498             std::string token = reader.next();
00499             if (token.empty()) break; //End of equation
00500
00501             if (token == "(") {
00502                 parentheses.push(token);
00503             } else if (token == ")") {
00504                 if (parentheses.empty() || parentheses.top() != "(") {
00505                     return false; // Unbalanced parentheses
00506                 }
00507                 parentheses.pop();
00508             } else if (is2ArgOperator(token)) {
00509                 operatorCount++;
00510             } else if (is_number(token)) {
00511                 operandCount++;
00512             } else if (!is1ArgOperator(token)) {
00513                 return false; // Invalid token
00514             }
00515         }
00516
00517         // Check if parentheses are balanced
00518         if (!parentheses.empty()) {
```

```
00519             return false;
00520         }
00521
00522         if (operandCount != operatorCount + 1) {
00523             return false;
00524         }
00525
00526         return true;
00527     }
00528
00529 }
00530
00531
```

6.15 lib/RPN.h File Reference

```
#include <string>
#include <unordered_set>
#include <sstream>
```

Include dependency graph for RPN.h: This graph shows which files directly or indirectly include this file:

Classes

- struct RPN::TokenReader
- struct RPN::RPNSolver
- struct RPN::NotationConverter
- struct RPN::NotationDeterminer
- struct RPN::Spacer
- struct RPN::EquationValidator

Namespaces

- namespace RPN

6.16 RPN.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <string>
00003 #include <unordered_set>
00004 #include <sstream>
00005
00006 namespace RPN {
00010     struct TokenReader {
00015         explicit TokenReader(const std::string& string);
00020         std::string getString();
00025         std::string next();
00031         std::string peek();
00036         bool finished() const;
00037     private:
00038         std::string string_;
00039         std::stringstream stream;
00040     };
00041
00045     struct RPNSolver {
00050         static double getResult(const std::string& equation);
00051     };
00052
00056     struct NotationConverter {
00062         static std::string infixToRPN(const std::string& infix);
00068         static std::string RPNtoInfix(const std::string& RPN);
00069     private:
```

```
00078         static std::string wrapInParentheses(const std::string& a, const std::string& b, const
      std::string& op);
00087         static std::string aopb(const std::string &a, const std::string &b, const std::string &op);
00088         static std::string onlyParentheses(const std::string &a);
00089     };
00090
00091     struct NotationDeterminer {
00096         static bool isRPN(const std::string& equation);
00101         static bool isInfix(const std::string& equation);
00102     };
00103
00104     struct Spacer {
00110         static std::string addSpacesAroundParentheses(const std::string& input);
00111
00115         static std::string removeSpacesAroundParentheses(const std::string& input);
00116
00120         static std::string addSpacesAroundOperators(const std::string& input);
00121
00125         static std::string removeSpacesAroundOperators(const std::string& input);
00126
00130         static std::string mergeSpaces(const std::string& input);
00131     };
00132
00133     struct EquationValidator {
00138         static bool isValidRPN(const std::string& equation);
00143         static bool isValidInfix(const std::string& equation);
00144     private:
00149         static bool is_number(const std::string& str);
00150     };
00151 }
```

6.17 main.cpp File Reference

```
#include <iostream>
#include <cstdio>
#include <fstream>
#include <string>
#include "RPN.h"
```

Include dependency graph for main.cpp:

Functions

- void help ()
- void setFlags (const std::string &flags)
- void errorInvalidEquation ()
- void solveForOutput (const std::string &sourceEquation, std::string &outputEquation, double &result)
- int main (const int argc, char ∗argv[ ])

Variables

- int inputFilePos = -1
- int outputFilePos = -1
- bool isInteractive = false
- bool isRPNOutput = false

6.17.1 Function Documentation

6.17.1.1 errorInvalidEquation()

```
void errorInvalidEquation ()
```

Definition at line 59 of file main.cpp.

6.17.1.2 help()

```
void help ()
```

Outputs help when executable has no parameters.

Definition at line 10 of file main.cpp.

6.17.1.3 main()

```
int main (
            const int argc,
            char * argv[])
```

Definition at line 86 of file main.cpp.

6.17.1.4 setFlags()

```
void setFlags (
            const std::string & flags)
```

Reads flags and configures their values.

Parameters

flags	

Definition at line 36 of file main.cpp.

6.17.1.5 solveForOutput()

```
void solveForOutput (
            const std::string & sourceEquation,
            std::string & outputEquation,
            double & result)
```

Definition at line 64 of file main.cpp.

6.17.2 Variable Documentation

6.17.2.1 inputFilePos

```
int inputFilePos = -1
```

Definition at line 27 of file main.cpp.

6.17.2.2 isInteractive

```
bool isInteractive = false
```

Definition at line 29 of file main.cpp.

6.17.2.3 isRPNOutput

```
bool isRPNOutput = false
```

Definition at line 30 of file main.cpp.

6.17.2.4 outputFilePos

```
int outputFilePos = -1
```

Definition at line 28 of file main.cpp.

6.18 main.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <cstdio>
00003 #include <fstream>
00004 #include <string>
00005 #include "RPN.h"
00006
00010 void help() {
00011     std::cout « "----------\n";
00012     std::cout « "Usage:\n";
00013     std::cout « "----------\n";
00014     std::cout « "-i    Input file\n";
00015     std::cout « "-o    Output file\n";
00016     std::cout « "-c    Interactive input\n";
00017     std::cout « "-r    Use Postfix in output\n";
00018     std::cout « "----------\n";
00019     std::cout « "Examples:\n";
00020     std::cout « "----------\n";
00021     std::cout « "-io input.txt output.txt\n";
00022     std::cout « "-oi output.txt input.txt\n";
00023     std::cout « "-cor interactive_rpn_output.txt\n";
00024     std::cout « "----------\n";
00025 }
00026
00027 int inputFilePos    = -1;
00028 int outputFilePos   = -1;
00029 bool isInteractive  = false;
00030 bool isRPNOutput    = false;
00031
00036 void setFlags(const std::string &flags) {
00037     int nonPositionalSkips = 0; //c and r don't have parameters so need to be reduced.
00038     for (int pos=1; pos<flags.length(); pos++) {
00039         switch (flags[pos]) {
00040             case 'i':
00041                 inputFilePos = pos - nonPositionalSkips;
00042                 break;
00043             case 'o':
00044                 outputFilePos = pos - nonPositionalSkips;
00045                 break;
00046             case 'c':
00047                 isInteractive = true;
00048                 nonPositionalSkips++;
00049                 break;
00050             case 'r':
00051                 isRPNOutput = true;
00052                 nonPositionalSkips++;
00053                 break;
```

```
00054             default: break;
00055         }
00056     }
00057 }
00058
00059 void errorInvalidEquation() {
00060     std::cerr « "[ERROR]: This equation is invalid.\n";
00061     exit(1);
00062 }
00063
00064 void solveForOutput(const std::string &sourceEquation, std::string &outputEquation, double &result) {
00065     const std::string spacedCopy = RPN::Spacer::addSpacesAroundParentheses(sourceEquation);
00066     std::string rpn;
00067     std::string infix;
00068     if (RPN::NotationDeterminer::isInfix(spacedCopy)) {
00069         infix = spacedCopy;
00070         if (!RPN::EquationValidator::isValidInfix(infix)) {
00071             errorInvalidEquation();
00072         }
00073         rpn = RPN::NotationConverter::infixToRPN(infix);
00074     } else {
00075         rpn = spacedCopy;
00076         if (!RPN::EquationValidator::isValidRPN(rpn)) {
00077             errorInvalidEquation();
00078         }
00079         infix = RPN::NotationConverter::RPNtoInfix(rpn);
00080     }
00081
00082     outputEquation = isRPNOutput ? rpn : infix;
00083     result = RPN::RPNSolver::getResult(rpn);
00084 }
00085
00086 int main(const int argc, char* argv[]) {
00087     if (argc < 2) {
00088         help();
00089         exit(0);
00090     }
00091     setFlags(argv[1]);
00092
00093     std::string strEquation;
00094     if (isInteractive) {
00095         printf("Enter the equation: ");
00096         std::getline(std::cin, strEquation);
00097     } else if (inputFilePos != -1) {
00098         std::ifstream file(argv[inputFilePos+1]);
00099         std::ostringstream buffer;
00100         buffer « file.rdbuf();
00101         strEquation = buffer.str();
00102     } else {
00103         std::cerr « "No input source! - Use interactive or file input.";
00104         exit(1);
00105     }
00106         double result;
00107         std::string outputEquation;
00108         solveForOutput(strEquation, outputEquation, result);
00109         outputEquation = RPN::Spacer::removeSpacesAroundParentheses(outputEquation);
00110         outputEquation = RPN::Spacer::mergeSpaces(outputEquation);
00111         std::cout « outputEquation « " = " «result«std::endl;
00112         if (outputFilePos != -1) {
00113             std::ofstream outputFile(argv[outputFilePos+1], std::ofstream::out);
00114             outputFile « outputEquation;
00115             outputFile « " = ";
00116             outputFile « result;
00117         }
00118     return 0;
00119 }
```

Index