Processing CSV Files

Fabian, Chakong, Shishir,Abdirahman, and Roshan

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

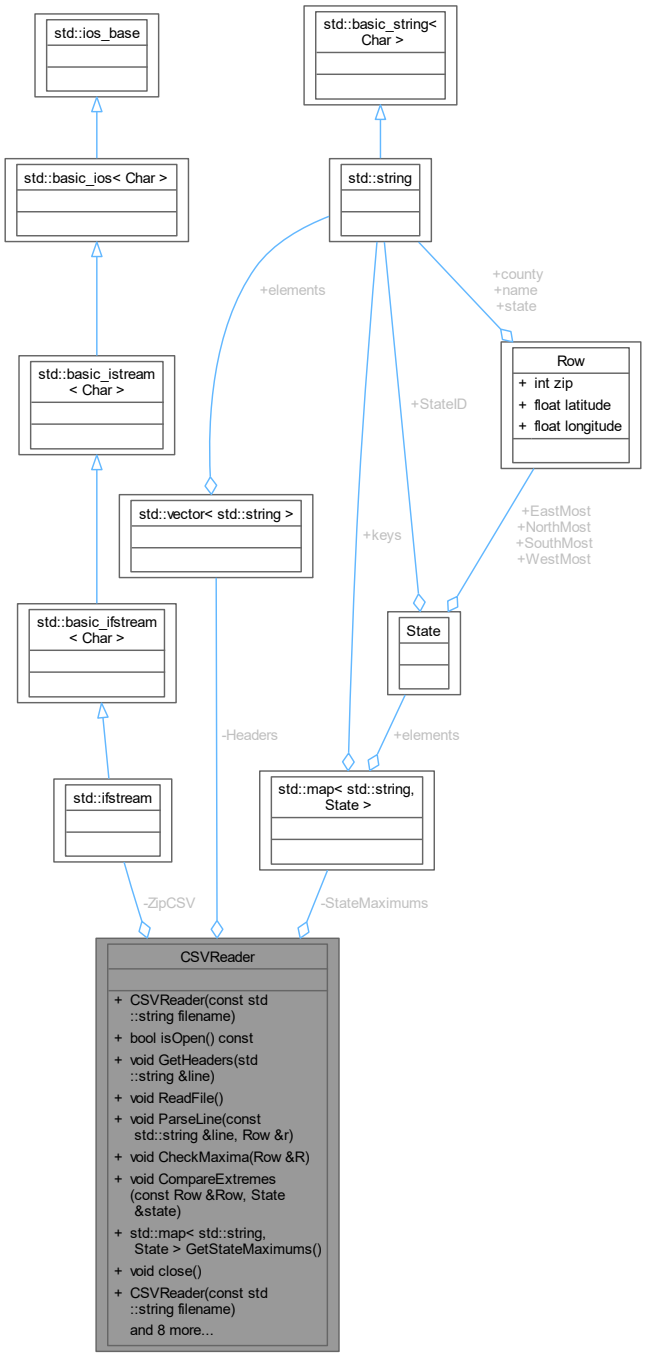
Chapter 3

Class Documentation

3.1 CSVReader Class Reference

```
#include <CSVReader.h>
```

Collaboration diagram for CSVReader:



Public Member Functions

- CSVReader (const std::string filename)

  Constructor that opens the CSV file specified by the 'filename' parameter.
- bool isOpen () const

  Checks if the CSV file is open.
- void GetHeaders (std::string &line)

Parses and stores the header row of the CSV file.

- void ReadFile ()

  Reads and processes the entire CSV file.
- void ParseLine (const std::string &line, Row &r)

  Parses a single data row of the CSV file into a Row object.
- void CheckMaxima (Row &R)

  Checks and updates StateMaximums map with maximum and minimum values.
- void CompareExtremes (const Row &Row, State &state)

  Compares and updates the maximum and minimum values for latitude and longitude in a State.
- std::map< std::string, State > GetStateMaximums ()

  Retrieves the StateMaximums map.
- void close ()

  Closes the CSV file if it's open.
- CSVReader (const std::string filename)

  Constructor that opens the CSV file specified by the 'filename' parameter.
- bool isOpen () const

  Checks if the CSV file is open.
- void GetHeaders (std::string &line)

  Parses and stores the header row of the CSV file.
- void ReadFile ()

  Reads and processes the entire CSV file.
- void ParseLine (const std::string &line, Row &r)

  Parses a single data row of the CSV file into a Row object.
- void CheckMaxima (Row &R)

  Checks and updates StateMaximums map with maximum and minimum values.
- void CompareExtremes (const Row &Row, State &state)

  Compares and updates the maximum and minimum values for latitude and longitude in a State.
- std::map< std::string, State > GetStateMaximums ()

  Retrieves the StateMaximums map.
- void close ()

  Closes the CSV file if it's open.

Private Attributes

- std::ifstream ZipCSV
- std::vector< std::string > Headers
- std::map< std::string, State > StateMaximums

3.1.1 Detailed Description

Definition at line 63 of file CSVReader.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CSVReader() [1/2]

```
CSVReader::CSVReader (
            const std::string filename )
```

Constructor that opens the CSV file specified by the 'filename' parameter.

Parameters

filename	The name of the CSV file to open.

Precondition

None.

Postcondition

The CSVReader object is constructed, and the CSV file is opened for reading.

Definition at line 45 of file CSVReader.cpp.

3.1.2.2 CSVReader() [2/2]

```
CSVReader::CSVReader (
            const std::string filename )
```

Constructor that opens the CSV file specified by the 'filename' parameter.

Parameters

filename	The name of the CSV file to open.

Precondition

None.

Postcondition

The CSVReader object is constructed, and the CSV file is opened for reading.

3.1.3 Member Function Documentation

3.1.3.1 CheckMaxima() [1/2]

```
void CSVReader::CheckMaxima (
            Row & R )
```

Checks and updates StateMaximums map with maximum and minimum values.

Parameters

R	Reference to the Row object to check and update the StateMaximums map.

Precondition

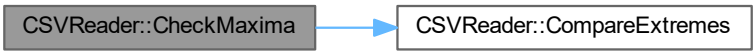
The CSV file is open for reading.

Postcondition

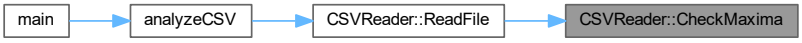
The StateMaximums map is updated with extremity values from the input 'R'.

Definition at line 149 of file CSVReader.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.2 CheckMaxima() [2/2]

```
void CSVReader::CheckMaxima (
            Row & R )
```

Checks and updates StateMaximums map with maximum and minimum values.

Parameters

R	Reference to the Row object to check and update the StateMaximums map.

Precondition

The CSV file is open for reading.

Postcondition

The StateMaximums map is updated with extremity values from the input 'R'.

3.1.3.3 close() [1/2]

```
void CSVReader::close ( )
```

Closes the CSV file if it's open.

Precondition

None.

Postcondition

The CSV file is closed if it was open.

Definition at line 232 of file CSVReader.cpp.

Here is the caller graph for this function:

3.1.3.4 close() [2/2]

```
void CSVReader::close ( )
```

Closes the CSV file if it's open.

Precondition

None.

Postcondition

The CSV file is closed if it was open.

3.1.3.5 CompareExtremes() [1/2]

```
void CSVReader::CompareExtremes (
            const Row & Row,
            State & state )
```

Compares and updates the maximum and minimum values for latitude and longitude in a State.

Compares and updates the maximum and minimum values for latitude and longitude in a State. States with identical lat or long default to Zip for final comparison.

Parameters

Row	Reference to the Row object to compare with the State's extremities.
state	Reference to the State object containing the extremities to be updated.

Precondition

The CSV file is open for reading.

Postcondition

The State object 'state' is updated with extremity values from the input 'Row'.

Definition at line 176 of file CSVReader.cpp.

Here is the caller graph for this function:

3.1.3.6 CompareExtremes() [2/2]

```
void CSVReader::CompareExtremes (
            const Row & Row,
            State & state )
```

Compares and updates the maximum and minimum values for latitude and longitude in a State.

Parameters

Row	Reference to the Row object to compare with the State's extremities.
state	Reference to the State object containing the extremities to be updated.

Precondition

The CSV file is open for reading.

Postcondition

The State object 'state' is updated with extremity values from the input 'Row'.

3.1.3.7 GetHeaders() [1/2]

```
void CSVReader::GetHeaders (
            std::string & line )
```

Parses and stores the header row of the CSV file.

Parameters

line	The header row of the CSV file.

Precondition

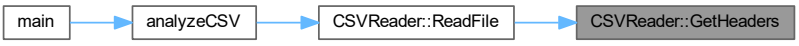
The CSV file is open for reading.

Postcondition

The 'Headers' vector is populated with column headers from the CSV file.

Definition at line 65 of file CSVReader.cpp.

Here is the caller graph for this function:



3.1.3.8 GetHeaders() [2/2]

```
void CSVReader::GetHeaders (
            std::string & line )
```

Parses and stores the header row of the CSV file.

Parameters

line	The header row of the CSV file.

Precondition

The CSV file is open for reading.

Postcondition

The 'Headers' vector is populated with column headers from the CSV file.

3.1.3.9 GetStateMaximums() [1/2]

```
std::map< std::string, State > CSVReader::GetStateMaximums ( )
```

Retrieves the StateMaximums map.

Returns

A copy of the StateMaximums map.

Precondition

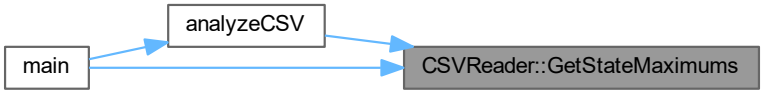
None.

Postcondition

None.

Definition at line 223 of file CSVReader.cpp.

Here is the caller graph for this function:



3.1.3.10 GetStateMaximums() [2/2]

```
std::map< std::string, State > CSVReader::GetStateMaximums ( )
```

Retrieves the StateMaximums map.

Returns

A copy of the StateMaximums map.

Precondition

None.

Postcondition

None.

3.1.3.11 isOpen() [1/2]

```
bool CSVReader::isOpen ( ) const
```

Checks if the CSV file is open.

Returns

true if the CSV file is open, false otherwise.

Precondition

None.

Postcondition

None.

Definition at line 55 of file CSVReader.cpp.

Here is the caller graph for this function:

3.1.3.12 isOpen() [2/2]

```
bool CSVReader::isOpen ( ) const
```

Checks if the CSV file is open.

Returns

true if the CSV file is open, false otherwise.

Precondition

None.

Postcondition

None.

3.1.3.13 ParseLine() [1/2]

```
void CSVReader::ParseLine (
            const std::string & Line,
            Row & r )
```

Parses a single data row of the CSV file into a Row object.

Parameters

Line	The data row to parse.
r	Reference to the Row object to store the parsed data.

Precondition

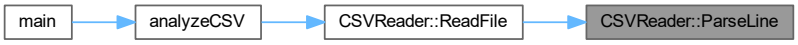
The CSV file is open for reading.

Postcondition

The 'r' object is updated with data from the input 'Line'.

Definition at line 102 of file CSVReader.cpp.

Here is the caller graph for this function:



3.1.3.14 ParseLine() [2/2]

```
void CSVReader::ParseLine (
            const std::string & line,
            Row & r )
```

Parses a single data row of the CSV file into a Row object.

Parameters

Line	The data row to parse.
r	Reference to the Row object to store the parsed data.

Precondition

The CSV file is open for reading.

Postcondition

The 'r' object is updated with data from the input 'Line'.

3.1.3.15 ReadFile() [1/2]

```
void CSVReader::ReadFile ( )
```

Reads and processes the entire CSV file.

Precondition

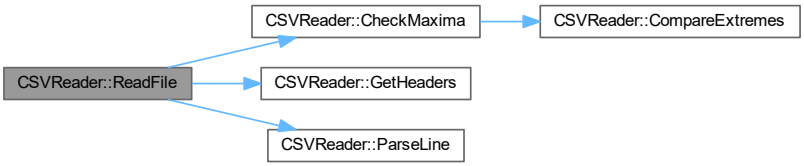
The CSV file is open for reading.

Postcondition

The CSV file is read, and data is parsed and stored in memory.

Definition at line 80 of file CSVReader.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.1.3.16 ReadFile() [2/2]

```
void CSVReader::ReadFile ( )
```

Reads and processes the entire CSV file.

Precondition

The CSV file is open for reading.

Postcondition

The CSV file is read, and data is parsed and stored in memory.

3.1.4 Member Data Documentation

3.1.4.1 Headers

```
std::vector< std::string > CSVReader::Headers  [private]
```

Stores the column headers from the CSV file.

Definition at line 141 of file CSVReader.h.

3.1.4.2 StateMaximums

```
std::map< std::string, State > CSVReader::StateMaximums  [private]
```

Stores state ID, as well as the maximum locations.

Definition at line 142 of file CSVReader.h.

3.1.4.3 ZipCSV

```
std::ifstream CSVReader::ZipCSV  [private]
```

Represents the input CSV file stream used to open and read the CSV file.

Definition at line 140 of file CSVReader.h.

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

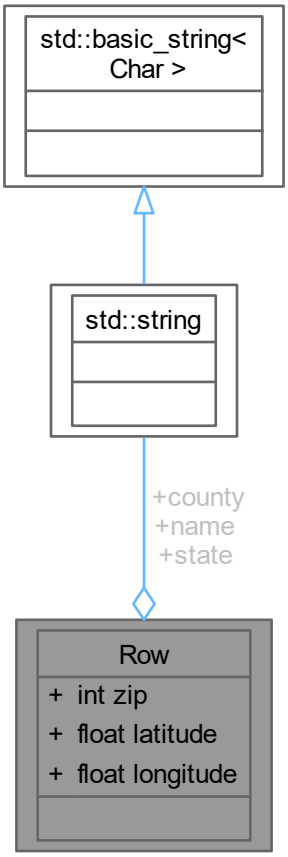
- CSVReader.h
- ZipCodes-main/CSVReader.h
- CSVReader.cpp
- ZipCodes-main/CSVReader.cpp

3.2 Row Struct Reference

Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude.

```
#include <CSVReader.h>
```

Collaboration diagram for Row:



Public Attributes

- int zip
- std::string name
- std::string state
- std::string county
- float latitude
- float longitude

3.2.1 Detailed Description

Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude.

Definition at line 40 of file CSVReader.h.

3.2.2 Member Data Documentation

3.2.2.1 county

```
std::string Row::county
```

The county.

Definition at line 44 of file CSVReader.h.

3.2.2.2 latitude

```
float Row::latitude
```

The latitude.

Definition at line 45 of file CSVReader.h.

3.2.2.3 longitude

```
float Row::longitude
```

The longitude.

Definition at line 46 of file CSVReader.h.

3.2.2.4 name

```
std::string Row::name
```

The place name.

Definition at line 42 of file CSVReader.h.

3.2.2.5 state

```
std::string Row::state
```

The state.

Definition at line 43 of file CSVReader.h.

3.2.2.6 zip

```
int Row::zip
```

The ZIP code.

Definition at line 41 of file CSVReader.h.

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

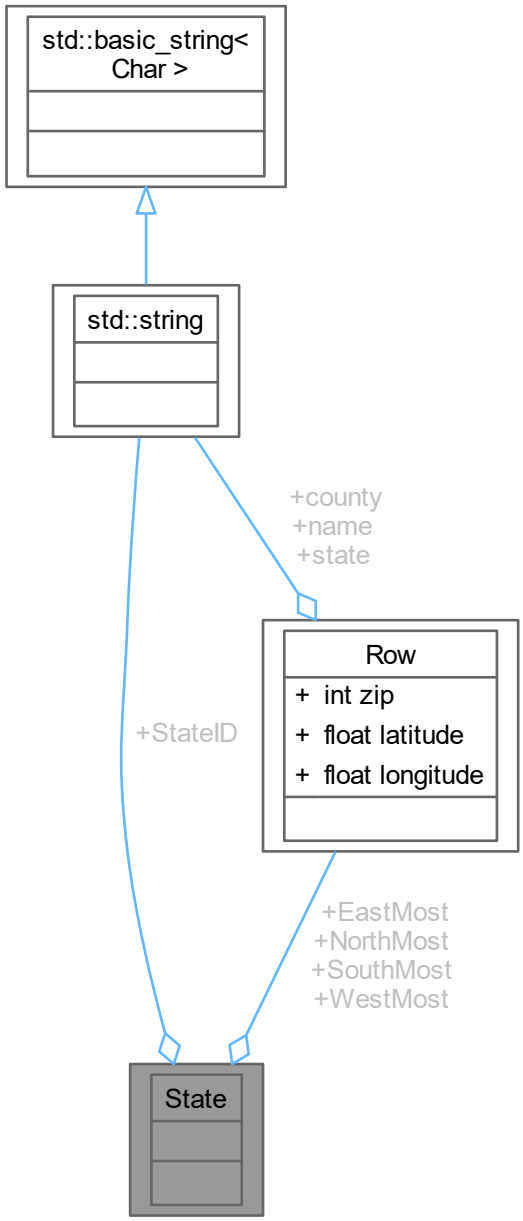
- CSVReader.h
- ZipCodes-main/CSVReader.h

3.3 State Struct Reference

Represents state-related data. This struct stores information related to a state, including the state ID, and the extreme values for latitude and longitude (NorthMost, SouthMost, EastMost, and WestMost rows) within that state.

`#include <CSVReader.h>`

Collaboration diagram for State:



Public Attributes

- std::string StateID
- Row NorthMost
- Row SouthMost
- Row EastMost
- Row WestMost

3.3.1 Detailed Description

Represents state-related data. This struct stores information related to a state, including the state ID, and the extreme values for latitude and longitude (NorthMost, SouthMost, EastMost, and WestMost rows) within that state.

Definition at line 55 of file CSVReader.h.

3.3.2 Member Data Documentation

3.3.2.1 EastMost

```
Row State::EastMost
```

The row with the easternmost longitude.

Definition at line 59 of file CSVReader.h.

3.3.2.2 NorthMost

```
Row State::NorthMost
```

The row with the northernmost latitude.

Definition at line 57 of file CSVReader.h.

3.3.2.3 SouthMost

```
Row State::SouthMost
```

The row with the southernmost latitude.

Definition at line 58 of file CSVReader.h.

3.3.2.4 StateID

```
std::string State::StateID
```

The state ID.

Definition at line 56 of file CSVReader.h.

3.3.2.5 WestMost

```
Row State::WestMost
```

The row with the westernmost longitude.

Definition at line 60 of file CSVReader.h.

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

- CSVReader.h
- ZipCodes-main/CSVReader.h

Chapter 4

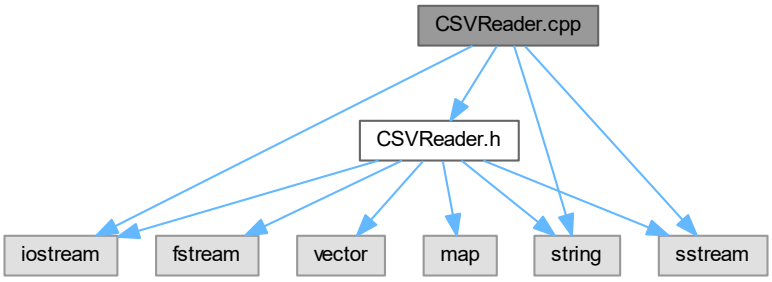
File Documentation

4.1 CSVReader.cpp File Reference

Member function definitions for class CSVReader.

```
#include "CSVReader.h"
#include <iostream>
#include <string>
#include <sstream>
```

Include dependency graph for CSVReader.cpp:



4.1.1 Detailed Description

Member function definitions for class CSVReader.

Author

Fabian MullerDahlberg

Authors

(comments by Hamaad) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CCSVReader.h for declaration.

- Constructor: Opens a specified CSV file for reading.
- isOpen(): Checks if the CSV file is currently open.
- GetHeaders(): Parses and stores the header row of the CSV file, populating the Headers vector with column headers.
- ReadFile(): Reads and processes the entire CSV file, including parsing data rows and calculating state statistics.
- ParseLine(): Parses a single data row of the CSV file into a Row object, updating it with data from the input line.
- CheckMaxima(): Checks and updates a map (StateMaximums) with maximum and minimum values for latitude and longitude based on the input Row.
- CompareExtremes(): Compares and updates the maximum and minimum values for latitude and longitude in a state.
- GetStateMaximums(): Retrieves a copy of the StateMaximums map, which contains state statistics.
- close(): Closes the CSV file if it's currently open.

Assumptions:

- The input CSV file is properly formatted with valid data.
- The CSV file has a header row that defines column names.
- Latitude and longitude values are provided in decimal format.
- The CSV file contains data for multiple states.
- The CSV file follows the format: Zip,Name,State,County,Latitude,Longitude.
- Rows with missing or invalid data will be skipped.
- The CSV file may be large, so memory usage is considered.
- State statistics, including maximum and minimum values, are calculated and stored for each state in the data.

Definition in file CSVReader.cpp.

4.2 CSVReader.cpp

Go to the documentation of this file.

```
00001
00034 #include "CSVReader.h"
00035 #include <iostream>
00036 #include <string>
00037 #include <sstream>
00038
00045 CSVReader::CSVReader(const std::string filename) {
00046     ZipCSV.open(filename, std::ios::in);
00047 }
00048
00055 bool CSVReader::isOpen() const {
00056     return ZipCSV.is_open();
00057 }
00058
```

```
00065 void CSVReader::GetHeaders(std::string &line) {
00066     std::stringstream stream(line);
00067     std::string header;
00068     // Parses the first of the csv by comma to get headers
00069     while (std::getline(stream, header, ',')) {
00070         // Adds headers to the vector
00071         Headers.push_back(header);
00072     }
00073 }
00074
00080 void CSVReader::ReadFile() {
00081     std::string line;
00082
00083     // Read and store the header row of the CSV file.
00084     std::getline(ZipCSV, line, '\n');
00085     GetHeaders(line);
00086
00087     // Read and process each data row of the CSV file.
00088     while (std::getline(ZipCSV, line, '\n')) {
00089         Row NewRow;
00090         ParseLine(line, NewRow);
00091         CheckMaxima(NewRow);
00092     }
00093 }
00094
00102 void CSVReader::ParseLine(const std::string& Line, Row& r) {
00103     std::stringstream stream(Line);
00104     std::string item;
00105     int fieldIndex = 0;
00106
00107     while (std::getline(stream, item, ',')) {
00108         switch (fieldIndex) {
00109             case 0: // Zip
00110                 try {
00111                     r.zip = std::stoi(item);
00112                 } catch (const std::invalid_argument&) {
00113                     // Not an integer
00114                 }break;
00115             case 1: // Name
00116                 r.name = item;
00117                 break;
00118             case 2: // State
00119                 r.state = item;
00120                 break;
00121             case 3: // County
00122                 r.county = item;
00123                 break;
00124             case 4: // Latitude
00125                 try {
00126                     r.latitude = std::stof(item);
00127                 } catch (const std::invalid_argument&) {
00128                     // Not a float
00129                 }break;
00130             case 5: // Longitude
00131                 try {
00132                     r.longitude = std::stof(item);
00133                 } catch (const std::invalid_argument&) {
00134                     // Not a float
00135                 }break;
00136             default:
00137                 break;
00138         }
00139         fieldIndex++;
00140     }
00141 }
00142
00149 void CSVReader::CheckMaxima(Row &R) {
00150     // If the state is already in the map
00151     if (!(StateMaximums.find(R.state) == StateMaximums.end())) {
00152         CompareExtremes(R, StateMaximums[R.state]);
00153     }
00154         // If the state is not in the map, create a new State and initialize it with the input Row.
00155     else {
00156         State newState;
00157         newState.StateID = R.state;
00158         newState.NorthMost = R;
00159         newState.SouthMost = R;
00160         newState.EastMost = R;
00161         newState.WestMost = R;
00162
00163         // Insert the new State into the map.
00164         StateMaximums[R.state] = newState;
00165     }
00166 }
00167
00176 void CSVReader::CompareExtremes(const Row &Row, State &state) {
00177
```

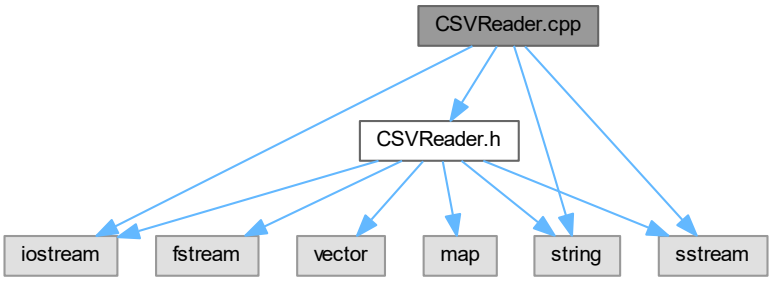
```
00178     // Compare latitude for northmost and southmost
00179     if (Row.latitude > state.NorthMost.latitude) {
00180         state.NorthMost = Row;
00181         // If latitude is the same the row with the largest zip is used
00182     }else if (Row.latitude == state.NorthMost.latitude) {
00183         if (Row.zip > state.NorthMost.zip) {
00184             state.NorthMost = Row;
00185         }
00186     }
00187
00188     if (Row.latitude < state.SouthMost.latitude) {
00189         state.SouthMost = Row;
00190         // If latitude is the same the row with the smallest zip is used
00191     }else if (Row.latitude == state.SouthMost.latitude) {
00192         if (Row.zip < state.SouthMost.zip) {
00193             state.SouthMost = Row;
00194         }
00195     }
00196
00197     // Compare longitude for eastmost and westmost
00198     if (Row.longitude > state.EastMost.longitude) {
00199         state.EastMost = Row;
00200         // If longitude is the same the row with the largest zip is used
00201     }else if (Row.longitude == state.EastMost.longitude) {
00202         if (Row.zip > state.EastMost.zip) {
00203             state.EastMost = Row;
00204         }
00205     }
00206
00207     if (Row.longitude < state.WestMost.longitude) {
00208         state.WestMost = Row;
00209         // If longitude is the same the row with the smallest zip is used
00210     }else if (Row.longitude == state.WestMost.longitude) {
00211         if (Row.zip < state.WestMost.zip) {
00212             state.WestMost = Row;
00213         }
00214     }
00215 }
00216
00223 std::map<std::string, State> CSVReader::GetStateMaximums() {
00224     return StateMaximums;
00225 }
00226
00232 void CSVReader::close() {
00233     if (ZipCSV.is_open()) {
00234         ZipCSV.close();
00235     }
00236 }
```

4.3 ZipCodes-main/CSVReader.cpp File Reference

Member function definitions for class CSVReader.

```
#include "CSVReader.h"
#include <iostream>
#include <string>
#include <sstream>
```

Include dependency graph for ZipCodes-main/CSVReader.cpp:



4.3.1 Detailed Description

Member function definitions for class CSVReader.

Author

Fabian MullerDahlberg

Authors

(comments by Hamaad) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CCSVReader.h for declaration.

- Constructor: Opens a specified CSV file for reading.
- isOpen(): Checks if the CSV file is currently open.
- GetHeaders(): Parses and stores the header row of the CSV file, populating the Headers vector with column headers.
- ReadFile(): Reads and processes the entire CSV file, including parsing data rows and calculating state statistics.
- ParseLine(): Parses a single data row of the CSV file into a Row object, updating it with data from the input line.
- CheckMaxima(): Checks and updates a map (StateMaximums) with maximum and minimum values for latitude and longitude based on the input Row.
- CompareExtremes(): Compares and updates the maximum and minimum values for latitude and longitude in a state.
- GetStateMaximums(): Retrieves a copy of the StateMaximums map, which contains state statistics.
- close(): Closes the CSV file if it's currently open.

Assumptions:

- The input CSV file is properly formatted with valid data.
- The CSV file has a header row that defines column names.
- Latitude and longitude values are provided in decimal format.
- The CSV file contains data for multiple states.
- The CSV file follows the format: Zip,Name,State,County,Latitude,Longitude.
- Rows with missing or invalid data will be skipped.
- The CSV file may be large, so memory usage is considered.
- State statistics, including maximum and minimum values, are calculated and stored for each state in the data.

Definition in file ZipCodes-main/CSVReader.cpp.

4.4 ZipCodes-main/CSVReader.cpp

Go to the documentation of this file.

```
00001
00034 #include "CSVReader.h"
00035 #include <iostream>
00036 #include <string>
00037 #include <sstream>
00038
00045 CSVReader::CSVReader(const std::string filename) {
00046     ZipCSV.open(filename, std::ios::in);
00047 }
00048
00055 bool CSVReader::isOpen() const {
00056     return ZipCSV.is_open();
00057 }
00058
00065 void CSVReader::GetHeaders(std::string &line) {
00066     std::stringstream stream(line);
00067     std::string header;
00068     // Parses the first of the csv by comma to get headers
00069     while (std::getline(stream, header, ',')) {
00070         // Adds headers to the vector
00071         Headers.push_back(header);
00072     }
00073 }
00074
00080 void CSVReader::ReadFile() {
00081     std::string line;
00082
00083     // Read and store the header row of the CSV file.
00084     std::getline(ZipCSV, line, '\n');
00085     GetHeaders(line);
00086
00087     // Read and process each data row of the CSV file.
00088     while (std::getline(ZipCSV, line, '\n')) {
00089         Row NewRow;
00090         ParseLine(line, NewRow);
00091         CheckMaxima(NewRow);
00092     }
00093 }
00094
00102 void CSVReader::ParseLine(const std::string& Line, Row& r) {
00103     std::stringstream stream(Line);
00104     std::string item;
00105     int fieldIndex = 0;
00106
00107     while (std::getline(stream, item, ',')) {
00108         switch (fieldIndex) {
00109             case 0: // Zip
00110                 try {
00111                     r.zip = std::stoi(item);
00112                 } catch (const std::invalid_argument&) {
00113                     // Not an integer
00114                 }break;
00115             case 1: // Name
00116                 r.name = item;
00117                 break;
00118             case 2: // State
00119                 r.state = item;
00120                 break;
00121             case 3: // County
00122                 r.county = item;
00123                 break;
00124             case 4: // Latitude
00125                 try {
00126                     r.latitude = std::stof(item);
00127                 } catch (const std::invalid_argument&) {
00128                     // Not a float
00129                 }break;
00130             case 5: // Longitude
00131                 try {
00132                     r.longitude = std::stof(item);
00133                 } catch (const std::invalid_argument&) {
00134                     // Not a float
00135                 }break;
00136             default:
00137                 break;
00138         }
00139         fieldIndex++;
00140     }
00141 }
00142
00149 void CSVReader::CheckMaxima(Row &R) {
00150     // If the state is already in the map
```

```
00151     if (!(StateMaximums.find(R.state) == StateMaximums.end())) {
00152         CompareExtremes(R, StateMaximums[R.state]);
00153     }
00154         // If the state is not in the map, create a new State and initialize it with the input Row.
00155     else {
00156         State newState;
00157         newState.StateID = R.state;
00158         newState.NorthMost = R;
00159         newState.SouthMost = R;
00160         newState.EastMost = R;
00161         newState.WestMost = R;
00162 
00163         // Insert the new State into the map.
00164         StateMaximums[R.state] = newState;
00165     }
00166 }
00167 
00176 void CSVReader::CompareExtremes(const Row &Row, State &state) {
00177 
00178     // Compare latitude for northmost and southmost
00179     if (Row.latitude > state.NorthMost.latitude) {
00180         state.NorthMost = Row;
00181         // If latitude is the same the row with the largest zip is used
00182     }else if (Row.latitude == state.NorthMost.latitude) {
00183         if (Row.zip > state.NorthMost.zip) {
00184             state.NorthMost = Row;
00185         }
00186     }
00187 
00188     if (Row.latitude < state.SouthMost.latitude) {
00189         state.SouthMost = Row;
00190         // If latitude is the same the row with the smallest zip is used
00191     }else if (Row.latitude == state.SouthMost.latitude) {
00192         if (Row.zip < state.SouthMost.zip) {
00193             state.SouthMost = Row;
00194         }
00195     }
00196 
00197     // Compare longitude for eastmost and westmost
00198     if (Row.longitude > state.EastMost.longitude) {
00199         state.EastMost = Row;
00200         // If longitude is the same the row with the largest zip is used
00201     }else if (Row.longitude == state.EastMost.longitude) {
00202         if (Row.zip > state.EastMost.zip) {
00203             state.EastMost = Row;
00204         }
00205     }
00206 
00207     if (Row.longitude < state.WestMost.longitude) {
00208         state.WestMost = Row;
00209         // If longitude is the same the row with the smallest zip is used
00210     }else if (Row.longitude == state.WestMost.longitude) {
00211         if (Row.zip < state.WestMost.zip) {
00212             state.WestMost = Row;
00213         }
00214     }
00215 }
00216 
00223 std::map<std::string, State> CSVReader::GetStateMaximums() {
00224     return StateMaximums;
00225 }
00226 
00232 void CSVReader::close() {
00233     if (ZipCSV.is_open()) {
00234         ZipCSV.close();
00235     }
00236 }
```

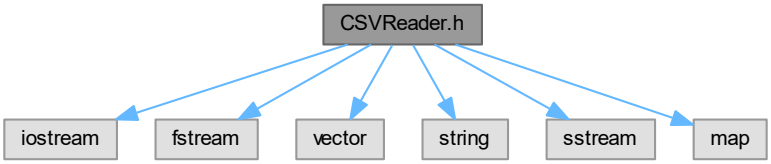
4.5 CSVReader.h File Reference

Declarations for class CSVReader.

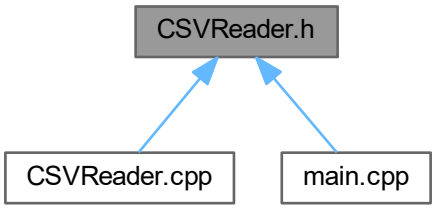
```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
```

```
#include <map>
```

Include dependency graph for CSVReader.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Row

  Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude.
- struct State

  Represents state-related data. This struct stores information related to a state, including the state ID, and the extreme values for latitude and longitude (NorthMost, SouthMost, EastMost, and WestMost rows) within that state.
- class CSVReader

4.5.1 Detailed Description

Declarations for class CSVReader.

Author

Fabian MullerDahlberg

(Comments by Roshan) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CSVReader.cpp for the implementation of these functions.

This file declares the class CSVReader, which provides functionality to read and process CSV files. The class includes member functions for opening, reading, and analyzing CSV files, as well as storing and retrieving state statistics.

Assumptions:

- The input CSV file is properly formatted with valid data.
- The CSV file has a header row that defines column names.
- Latitude and longitude values are provided in decimal format.
- The CSV file contains data for multiple states.
- The CSV file follows the format: Zip,Name,State,County,Latitude,Longitude.
- Rows with missing or invalid data will be skipped.
- The CSV file may be large, so memory usage is considered.
- State statistics, including maximum and minimum values, are calculated and stored for each state in the data.

Definition in file CSVReader.h.

4.6 CSVReader.h

Go to the documentation of this file.

```
00001
00024 #ifndef ZIPCODES_CSVREADER_H
00025 #define ZIPCODES_CSVREADER_H
00026
00027
00028 #include <iostream>
00029 #include <fstream>
00030 #include <vector>
00031 #include <string>
00032 #include <sstream>
00033 #include <map>
00034
00040 struct Row {
00041     int zip;
00042     std::string name;
00043     std::string state;
00044     std::string county;
00045     float latitude;
00046     float longitude;
00047 };
00048
00055 struct State {
00056     std::string StateID;
00057     Row NorthMost;
00058     Row SouthMost;
00059     Row EastMost;
00060     Row WestMost;
00061 };
00062
00063 class CSVReader {
00064 public:
00065
00072     CSVReader(const std::string filename);
00073
00080     bool isOpen() const;
00081
00088     void GetHeaders(std::string &line);
00089
00095     void ReadFile();
```

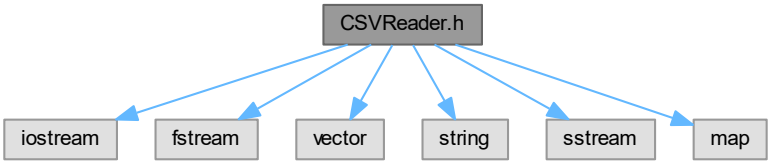
```
00096
00104     void ParseLine(const std::string &line, Row& r);
00105
00112     void CheckMaxima(Row &R);
00113
00122     void CompareExtremes(const Row &Row, State& state);
00123
00130     std::map<std::string, State> GetStateMaximums();
00131
00137     void close();
00138
00139 private:
00140     std::ifstream ZipCSV;
00141     std::vector<std::string> Headers;
00142     std::map<std::string, State> StateMaximums;
00143 };
00144
00145 #endif //ZIPCODES_CSVREADER_H
```

4.7 ZipCodes-main/CSVReader.h File Reference

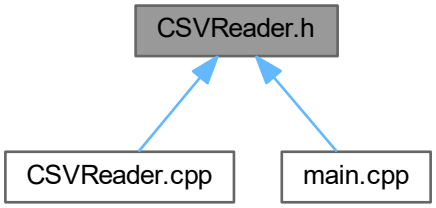
Declarations for class CSVReader.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <map>
```

Include dependency graph for ZipCodes-main/CSVReader.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Row

  Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude.
- struct State

  Represents state-related data. This struct stores information related to a state, including the state ID, and the extreme values for latitude and longitude (NorthMost, SouthMost, EastMost, and WestMost rows) within that state.
- class CSVReader

4.7.1 Detailed Description

Declarations for class CSVReader.

Author

Fabian MullerDahlberg

(Comments by Roshan) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CSVReader.cpp for the implementation of these functions.

This file declares the class CSVReader, which provides functionality to read and process CSV files. The class includes member functions for opening, reading, and analyzing CSV files, as well as storing and retrieving state statistics.

Assumptions:

- The input CSV file is properly formatted with valid data.
- The CSV file has a header row that defines column names.
- Latitude and longitude values are provided in decimal format.
- The CSV file contains data for multiple states.
- The CSV file follows the format: Zip,Name,State,County,Latitude,Longitude.
- Rows with missing or invalid data will be skipped.
- The CSV file may be large, so memory usage is considered.
- State statistics, including maximum and minimum values, are calculated and stored for each state in the data.

Definition in file ZipCodes-main/CSVReader.h.

4.8 ZipCodes-main/CSVReader.h

Go to the documentation of this file.

```
00001
00024 #ifndef ZIPCODES_CSVREADER_H
00025 #define ZIPCODES_CSVREADER_H
00026
00027
00028 #include <iostream>
00029 #include <fstream>
00030 #include <vector>
00031 #include <string>
00032 #include <sstream>
00033 #include <map>
00034
00040 struct Row {
00041     int zip;
00042     std::string name;
00043     std::string state;
00044     std::string county;
00045     float latitude;
00046     float longitude;
00047 };
00048
00055 struct State {
00056     std::string StateID;
00057     Row NorthMost;
00058     Row SouthMost;
00059     Row EastMost;
00060     Row WestMost;
00061 };
00062
00063 class CSVReader {
00064 public:
00065
00072     CSVReader(const std::string filename);
00073
00080     bool isOpen() const;
00081
00088     void GetHeaders(std::string &line);
00089
00095     void ReadFile();
00096
00104     void ParseLine(const std::string &line, Row& r);
00105
00112     void CheckMaxima(Row &R);
00113
00122     void CompareExtremes(const Row &Row, State& state);
00123
00130     std::map<std::string, State> GetStateMaximums();
00131
00137     void close();
00138
00139 private:
00140     std::ifstream ZipCSV;
00141     std::vector<std::string> Headers;
00142     std::map<std::string, State> StateMaximums;
00143 };
00144
00145 #endif //ZIPCODES_CSVREADER_H
```

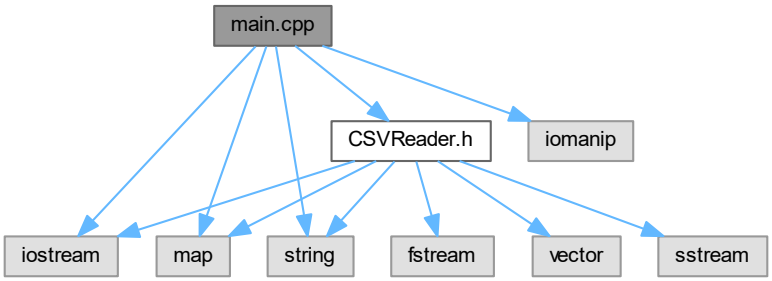
4.9 main.cpp File Reference

This program reads a CSV file containing postal code data, calculates state statistics, and displays the easternmost, westernmost, northernmost, and southernmost locations for each state.

```
#include <iostream>
#include <map>
#include <string>
#include <iomanip>
```

```
#include "CSVReader.h"
```

Include dependency graph for main.cpp:



Functions

- void analyzeCSV (CSVReader &csvReader)

  Analyzes and displays state statistics from a CSVReader object.
- bool AreStateMaximumsEqual (const std::map< std::string, State > &map1, const std::map< std::string, State > &map2)

  Compares two StateMaximums maps to check if they are equal.
- int main ()

  Main function to process and display state statistics from a CSV file.

4.9.1 Detailed Description

This program reads a CSV file containing postal code data, calculates state statistics, and displays the easternmost, westernmost, northernmost, and southernmost locations for each state.

Author

Chakong Lor

(Comments by Roshan and Fabian) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CCSVReader.h and CCSVReader.cpp for Class declaration, implementation, and Assumptions.

The program utilizes the CSVReader class to process the CSV file. The methods are run twice on two different csv's. One contains the rows ordered by zip code, smallest to largest, The other csv is ordered by location name alphabetically A-Z. The two running's are compared to ensure that their output is the same.

Definition in file main.cpp.

4.9.2 Function Documentation

4.9.2.1 analyzeCSV()

```
void analyzeCSV (
            CSVReader & csvReader )
```

Analyzes and displays state statistics from a CSVReader object.

Parameters

csvReader	The CSVReader object to analyze.

Precondition

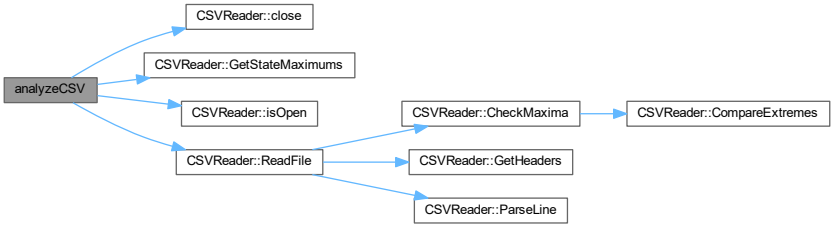
The CSVReader object is open and initialized.

Postcondition

State statistics are displayed for the given CSVReader object.

Definition at line 62 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.9.2.2 AreStateMaximumsEqual()

```
bool AreStateMaximumsEqual (
            const std::map< std::string, State > & map1,
            const std::map< std::string, State > & map2 )
```

Compares two StateMaximums maps to check if they are equal.

Parameters

map1	The first StateMaximums map.
map2	The second StateMaximums map.

Returns

True if the maps are equal, false otherwise.

Definition at line 94 of file main.cpp.

Here is the caller graph for this function:

4.9.2.3 main()

```
int main ( )
```

Main function to process and display state statistics from a CSV file.

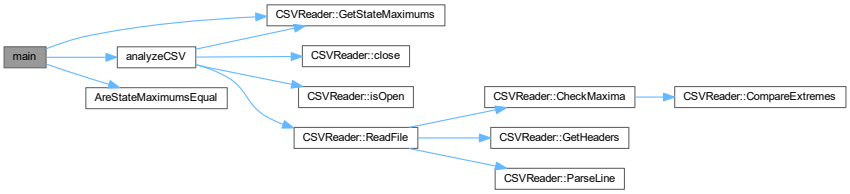
This function creates a CSVReader object, opens a CSV file, reads and processes the data, and displays state statistics.

Returns

0 on success, 1 on failure (e.g., if the CSV file cannot be opened).

Definition at line 33 of file main.cpp.

Here is the call graph for this function:



4.10 main.cpp

Go to the documentation of this file.

```
00001
00015 #include <iostream>
00016 #include <map>
00017 #include <string>
00018 #include <iomanip>
00019 #include "CSVReader.h"
00020
00021 // Declaration for analyzeCSV
00022 void analyzeCSV(CSVReader &file);
00023
00024 // Declaration for AreStateMaximumsEqual
00025 bool AreStateMaximumsEqual(const std::map<std::string, State>& map1, const std::map<std::string,
      State>& map2);
00026
00033 int main() {
00034     // Create a CSVReader object and open a CSV file
00035     std::string file = "us_postal_codes.csv";
00036     std::cout « "Processing us_postal_codes.csv. \n" « std::endl;
00037     CSVReader csvReader(file);
00038     analyzeCSV(csvReader);
00039
00040     std::string file2 = "us_postal_codes_place.csv";
00041     std::cout « "Processing us_postal_codes_place.csv. \n" « std::endl;
00042     CSVReader csvReader2(file2);
00043     analyzeCSV(csvReader2);
00044
00045     std::cout « "\n" « std::endl;
00046
00047     if (AreStateMaximumsEqual(csvReader.GetStateMaximums(), csvReader2.GetStateMaximums())) {
00048         std::cout « "StateMaximums maps are the same." « std::endl;
00049     } else {
00050         std::cout « "StateMaximums maps are different." « std::endl;
00051     }
00052
00053     return 0;
00054 }
00055
00062 void analyzeCSV(CSVReader &csvReader) {
00063     //CSVReader csvReader(fileName);
00064     if (!csvReader.isOpen()) {
00065         std::cerr « "Failed to open CSV file." « std::endl;
00066         return;
00067     }
00068     // Read and process the CSV file.
00069     csvReader.ReadFile();
00070
00071     // Print the header
00072     std::cout « "State       |  Eastmost        |  Westmost        |  Northmost       |  Southmost " «
      std::endl;
00073     csvReader.GetStateMaximums();
00074     // Iterate through the StateMaximums map and print statistics for each state
00075     for (const auto& pair : csvReader.GetStateMaximums() ) {
00076         const State& state = pair.second;
00077         std::cout « std::left « std::setw(10) « pair.first « "  |  "
00078                   « std::setw(14) « state.EastMost.zip « "  |  "
00079                   « std::setw(14) « state.WestMost.zip « "  |  "
00080                   « std::setw(14) « state.NorthMost.zip « "  |  "
00081                   « std::setw(14) « state.SouthMost.zip « std::endl;
00082     }
00083
00084     // Close the CSV file.
00085     csvReader.close();
00086 }
00087
00094 bool AreStateMaximumsEqual(const std::map<std::string, State>& map1, const std::map<std::string,
      State>& map2) {
00095     if (map1.size() != map2.size()) {
00096         return false; // Maps have different sizes, so they can't be the same.
00097     }
00098
00099     for (const auto& pair : map1) {
00100         const std::string& stateID = pair.first;
00101         const State& state1 = pair.second;
00102
00103         // Check if the stateID exists in map2.
00104         auto it = map2.find(stateID);
00105         if (it == map2.end()) {
00106             return false; // StateID not found in map2.
00107         }
00108
00109         const State& state2 = it->second;
00110
```

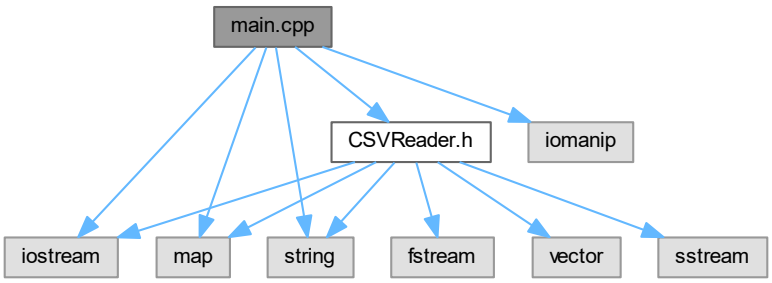
```
00111         // Compare the state data.
00112         if (state1.StateID != state2.StateID ||
00113             state1.NorthMost.zip != state2.NorthMost.zip ||
00114             state1.SouthMost.zip != state2.SouthMost.zip ||
00115             state1.EastMost.zip != state2.EastMost.zip ||
00116             state1.WestMost.zip != state2.WestMost.zip) {
00117             return false; // State data is different.
00118         }
00119     }
00120
00121     return true; // Maps are the same.
00122 }
```

4.11 ZipCodes-main/main.cpp File Reference

This program reads a CSV file containing postal code data, calculates state statistics, and displays the easternmost, westernmost, northernmost, and southernmost locations for each state.

```
#include <iostream>
#include <map>
#include <string>
#include <iomanip>
#include "CSVReader.h"
```

Include dependency graph for ZipCodes-main/main.cpp:



Functions

- void analyzeCSV (CSVReader &csvReader)

  Analyzes and displays state statistics from a CSVReader object.
- bool AreStateMaximumsEqual (const std::map< std::string, State > &map1, const std::map< std::string, State > &map2)

  Compares two StateMaximums maps to check if they are equal.
- int main ()

  Main function to process and display state statistics from a CSV file.

4.11.1 Detailed Description

This program reads a CSV file containing postal code data, calculates state statistics, and displays the easternmost, westernmost, northernmost, and southernmost locations for each state.

Author

Chakong Lor

(Comments by Roshan and Fabian) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CCSVReader.h and CCSVReader.cpp for Class declaration, implementation, and Assumptions.

The program utilizes the CSVReader class to process the CSV file. The methods are run twice on two different csv's. One contains the rows ordered by zip code, smallest to largest, The other csv is ordered by location name alphabetically A-Z. The two running's are compared to ensure that their output is the same.

Definition in file ZipCodes-main/main.cpp.

4.11.2 Function Documentation

4.11.2.1 analyzeCSV()

```
void analyzeCSV (
            CSVReader & csvReader )
```

Analyzes and displays state statistics from a CSVReader object.

Parameters

csvReader	The CSVReader object to analyze.

Precondition

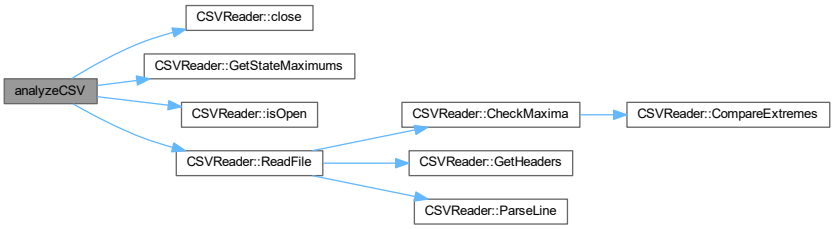
The CSVReader object is open and initialized.

Postcondition

State statistics are displayed for the given CSVReader object.

Definition at line 62 of file ZipCodes-main/main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.11.2.2 AreStateMaximumsEqual()

```
bool AreStateMaximumsEqual (
            const std::map< std::string, State > & map1,
            const std::map< std::string, State > & map2 )
```

Compares two StateMaximums maps to check if they are equal.

Parameters

map1	The first StateMaximums map.
map2	The second StateMaximums map.

Returns

True if the maps are equal, false otherwise.

Definition at line 94 of file ZipCodes-main/main.cpp.

Here is the caller graph for this function:

4.11.2.3 main()

```
int main ( )
```

Main function to process and display state statistics from a CSV file.

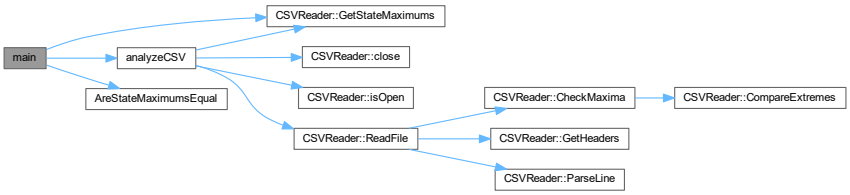
This function creates a CSVReader object, opens a CSV file, reads and processes the data, and displays state statistics.

Returns

0 on success, 1 on failure (e.g., if the CSV file cannot be opened).

Definition at line 33 of file ZipCodes-main/main.cpp.

Here is the call graph for this function:



4.12 ZipCodes-main/main.cpp

Go to the documentation of this file.

```
00001
00015 #include <iostream>
00016 #include <map>
00017 #include <string>
00018 #include <iomanip>
00019 #include "CSVReader.h"
00020
00021 // Declaration for analyzeCSV
00022 void analyzeCSV(CSVReader &file);
00023
00024 // Declaration for AreStateMaximumsEqual
00025 bool AreStateMaximumsEqual(const std::map<std::string, State>& map1, const std::map<std::string,
      State>& map2);
00026
00033 int main() {
00034     // Create a CSVReader object and open a CSV file
00035     std::string file = "us_postal_codes.csv";
00036     std::cout « "Processing us_postal_codes.csv. \n" « std::endl;
00037     CSVReader csvReader(file);
00038     analyzeCSV(csvReader);
00039
00040     std::string file2 = "us_postal_codes_place.csv";
00041     std::cout « "Processing us_postal_codes_place.csv. \n" « std::endl;
00042     CSVReader csvReader2(file2);
00043     analyzeCSV(csvReader2);
00044
00045     std::cout « "\n" « std::endl;
00046
00047     if (AreStateMaximumsEqual(csvReader.GetStateMaximums(), csvReader2.GetStateMaximums())) {
00048         std::cout « "StateMaximums maps are the same." « std::endl;
00049     } else {
00050         std::cout « "StateMaximums maps are different." « std::endl;
00051     }
00052
00053     return 0;
00054 }
00055
00062 void analyzeCSV(CSVReader &csvReader) {
00063     //CSVReader csvReader(fileName);
00064     if (!csvReader.isOpen()) {
00065         std::cerr « "Failed to open CSV file." « std::endl;
00066         return;
00067     }
00068     // Read and process the CSV file.
00069     csvReader.ReadFile();
00070
00071     // Print the header
00072     std::cout « "State       |  Eastmost        |  Westmost        |  Northmost       |  Southmost " «
      std::endl;
00073     csvReader.GetStateMaximums();
00074     // Iterate through the StateMaximums map and print statistics for each state
```

```
00075     for (const auto& pair : csvReader.GetStateMaximums() ) {
00076         const State& state = pair.second;
00077         std::cout « std::left « std::setw(10) « pair.first « "  |  "
00078                  « std::setw(14) « state.EastMost.zip « "  |  "
00079                  « std::setw(14) « state.WestMost.zip « "  |  "
00080                  « std::setw(14) « state.NorthMost.zip « "  |  "
00081                  « std::setw(14) « state.SouthMost.zip « std::endl;
00082     }
00083 
00084     // Close the CSV file.
00085     csvReader.close();
00086 }
00087 
00094 bool AreStateMaximumsEqual(const std::map<std::string, State>& map1, const std::map<std::string,
      State>& map2) {
00095     if (map1.size() != map2.size()) {
00096         return false; // Maps have different sizes, so they can't be the same.
00097     }
00098 
00099     for (const auto& pair : map1) {
00100         const std::string& stateID = pair.first;
00101         const State& state1 = pair.second;
00102 
00103         // Check if the stateID exists in map2.
00104         auto it = map2.find(stateID);
00105         if (it == map2.end()) {
00106             return false; // StateID not found in map2.
00107         }
00108 
00109         const State& state2 = it->second;
00110 
00111         // Compare the state data.
00112         if (state1.StateID != state2.StateID ||
00113             state1.NorthMost.zip != state2.NorthMost.zip ||
00114             state1.SouthMost.zip != state2.SouthMost.zip ||
00115             state1.EastMost.zip != state2.EastMost.zip ||
00116             state1.WestMost.zip != state2.WestMost.zip) {
00117             return false; // State data is different.
00118         }
00119     }
00120 
00121     return true; // Maps are the same.
00122 }
```

4.13 CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #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

4.13.1 Macro Definition Documentation

4.13.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

4.13.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 716 of file CMakeCCompilerId.c.

4.13.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 805 of file CMakeCCompilerId.c.

4.13.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 427 of file CMakeCCompilerId.c.

4.13.1.5 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 720 of file CMakeCCompilerId.c.

4.13.1.6 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 731 of file CMakeCCompilerId.c.

4.13.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 558 of file CMakeCCompilerId.c.

4.13.1.8 STRINGIFY

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

Definition at line 448 of file CMakeCCompilerId.c.

4.13.1.9 STRINGIFY_HELPER

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

Definition at line 447 of file CMakeCCompilerId.c.

4.13.2 Function Documentation

4.13.2.1 main()

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

Definition at line 839 of file CMakeCCompilerId.c.

4.13.3 Variable Documentation

4.13.3.1 info_arch

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

Definition at line 797 of file CMakeCCompilerId.c.

4.13.3.2 info_compiler

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

Definition at line 434 of file CMakeCCompilerId.c.

4.13.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

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



  "OFF"

"]"
```

Definition at line 821 of file CMakeCCompilerId.c.

4.13.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

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

Definition at line 818 of file CMakeCCompilerId.c.

4.13.3.5 info_platform

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

Definition at line 796 of file CMakeCCompilerId.c.

4.14 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(_CRAYC)
00228 # define COMPILER_ID "Cray"
00229 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00230 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00231
00232 #elif defined(__TI_COMPILER_VERSION__)
00233 # define COMPILER_ID "TI"
00234   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00235 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00236 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00237 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00238
00239 #elif defined(__CLANG_FUJITSU)
00240 # define COMPILER_ID "FujitsuClang"
00241 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00242 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00243 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00244 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00245
00246
00247 #elif defined(__FUJITSU)
00248 # define COMPILER_ID "Fujitsu"
00249 # if defined(__FCC_version__)
00250 #   define COMPILER_VERSION __FCC_version__
00251 # elif defined(__FCC_major__)
00252 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00253 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00254 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00255 # endif
00256 # if defined(__fcc_version)
00257 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00258 # elif defined(__FCC_VERSION)
00259 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00260 # endif
00261
00262
00263 #elif defined(__ghs__)
00264 # define COMPILER_ID "GHS"
00265 /* __GHS_VERSION_NUMBER = VVVVRP */
00266 # ifdef __GHS_VERSION_NUMBER
00267 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00268 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00269 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00270 # endif
00271
00272 #elif defined(__TASKING__)
00273 # define COMPILER_ID "Tasking"
00274   # define COMPILER_VERSION_MAJOR DEC(__VERSION__/1000)
00275   # define COMPILER_VERSION_MINOR DEC(__VERSION__ % 100)
00276 # define COMPILER_VERSION_INTERNAL DEC(__VERSION__)
00277
00278 #elif defined(__TINYC__)
00279 # define COMPILER_ID "TinyCC"
00280
00281 #elif defined(__BCC__)
00282 # define COMPILER_ID "Bruce"
00283
00284 #elif defined(__SCO_VERSION__)
00285 # define COMPILER_ID "SCO"
00286
00287 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00288 # define COMPILER_ID "ARMCC"
00289 #if __ARMCC_VERSION >= 1000000
00290   /* __ARMCC_VERSION = VRRPPPP */
```

```
00291   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00292   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00293   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00294 #else
00295   /* __ARMCC_VERSION = VRPPPP */
00296   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00297   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00298   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00299 #endif
00300
00301
00302 #elif defined(__clang__) && defined(__apple_build_version__)
00303 # define COMPILER_ID "AppleClang"
00304 # if defined(_MSC_VER)
00305 #  define SIMULATE_ID "MSVC"
00306 # endif
00307 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00308 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00309 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00310 # if defined(_MSC_VER)
00311   /* _MSC_VER = VVRR */
00312 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00313 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00314 # endif
00315 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00316
00317 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00318 # define COMPILER_ID "ARMClang"
00319   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00320   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00321   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION     % 10000)
00322 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00323
00324 #elif defined(__clang__)
00325 # define COMPILER_ID "Clang"
00326 # if defined(_MSC_VER)
00327 #  define SIMULATE_ID "MSVC"
00328 # endif
00329 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00330 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00331 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00332 # if defined(_MSC_VER)
00333   /* _MSC_VER = VVRR */
00334 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00335 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00336 # endif
00337
00338 #elif defined(__LCC__) && (defined(__GNUC__) || defined(__GNUG__) || defined(__MCST__))
00339 # define COMPILER_ID "LCC"
00340 # define COMPILER_VERSION_MAJOR DEC(__LCC__ / 100)
00341 # define COMPILER_VERSION_MINOR DEC(__LCC__ % 100)
00342 # if defined(__LCC_MINOR__)
00343 #  define COMPILER_VERSION_PATCH DEC(__LCC_MINOR__)
00344 # endif
00345 # if defined(__GNUC__) && defined(__GNUC_MINOR__)
00346 #  define SIMULATE_ID "GNU"
00347 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00348 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00349 #  if defined(__GNUC_PATCHLEVEL__)
00350 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00351 #  endif
00352 # endif
00353
00354 #elif defined(__GNUC__)
00355 # define COMPILER_ID "GNU"
00356 # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00357 # if defined(__GNUC_MINOR__)
00358 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00359 # endif
00360 # if defined(__GNUC_PATCHLEVEL__)
00361 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00362 # endif
00363
00364 #elif defined(_MSC_VER)
00365 # define COMPILER_ID "MSVC"
00366   /* _MSC_VER = VVRR */
00367 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00368 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00369 # if defined(_MSC_FULL_VER)
00370 #  if _MSC_VER >= 1400
00371     /* _MSC_FULL_VER = VVRRPPPPP */
00372 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00373 #  else
00374     /* _MSC_FULL_VER = VVRRPPPP */
00375 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00376 #  endif
00377 # endif
```

```
00378 # if defined(_MSC_BUILD)
00379 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00380 # endif
00381
00382 #elif defined(_ADI_COMPILER)
00383 # define COMPILER_ID "ADSP"
00384 #if defined(__VERSIONNUM__)
00385   /* __VERSIONNUM__ = 0xVVRRPPTT */
00386 #  define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ » 24 & 0xFF)
00387 #  define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ » 16 & 0xFF)
00388 #  define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ » 8 & 0xFF)
00389 #  define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00390 #endif
00391
00392 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00393 # define COMPILER_ID "IAR"
00394 # if defined(__VER__) && defined(__ICCARM__)
00395 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00396 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00397 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00398 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00399 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
      defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
      defined(__ICC8051__) || defined(__ICCSTM8__))
00400 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00401 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00402 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00403 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00404 # endif
00405
00406 #elif defined(__SDCC_VERSION_MAJOR) || defined(SDCC)
00407 # define COMPILER_ID "SDCC"
00408 # if defined(__SDCC_VERSION_MAJOR)
00409 #  define COMPILER_VERSION_MAJOR DEC(__SDCC_VERSION_MAJOR)
00410 #  define COMPILER_VERSION_MINOR DEC(__SDCC_VERSION_MINOR)
00411 #  define COMPILER_VERSION_PATCH DEC(__SDCC_VERSION_PATCH)
00412 # else
00413   /* SDCC = VRP */
00414 #  define COMPILER_VERSION_MAJOR DEC(SDCC/100)
00415 #  define COMPILER_VERSION_MINOR DEC(SDCC/10 % 10)
00416 #  define COMPILER_VERSION_PATCH DEC(SDCC    % 10)
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(__TI_COMPILER_VERSION__)
00669 # if defined(__TI_ARM__)
00670 #  define ARCHITECTURE_ID "ARM"
00671
00672 # elif defined(__MSP430__)
00673 #  define ARCHITECTURE_ID "MSP430"
00674
00675 # elif defined(__TMS320C28XX__)
00676 #  define ARCHITECTURE_ID "TMS320C28x"
00677
00678 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00679 #  define ARCHITECTURE_ID "TMS320C6x"
00680
00681 # else /* unknown architecture */
00682 #  define ARCHITECTURE_ID ""
00683 # endif
00684
00685 # elif defined(__ADSPSHARC__)
00686 #  define ARCHITECTURE_ID "SHARC"
00687
00688 # elif defined(__ADSPBLACKFIN__)
00689 #  define ARCHITECTURE_ID "Blackfin"
00690
00691 #elif defined(__TASKING__)
00692
00693 # if defined(__CTC__) || defined(__CPTC__)
00694 #  define ARCHITECTURE_ID "TriCore"
00695
00696 # elif defined(__CMCS__)
00697 #  define ARCHITECTURE_ID "MCS"
00698
00699 # elif defined(__CARM__)
00700 #  define ARCHITECTURE_ID "ARM"
00701
00702 # elif defined(__CARC__)
00703 #  define ARCHITECTURE_ID "ARC"
00704
00705 # elif defined(__C51__)
00706 #  define ARCHITECTURE_ID "8051"
00707
00708 # elif defined(__CPCP__)
00709 #  define ARCHITECTURE_ID "PCP"
00710
00711 # else
00712 #  define ARCHITECTURE_ID ""
00713 # endif
00714
00715 #else
00716 #  define ARCHITECTURE_ID
00717 #endif
00718
00719 /* Convert integer to decimal digit literals.  */
00720 #define DEC(n)                   \
00721   ('0' + (((n) / 10000000)%10)), \
00722   ('0' + (((n) / 1000000)%10)),  \
00723   ('0' + (((n) / 100000)%10)),   \
```

```
00724   ('0' + (((n) / 10000)%10)),     \
00725   ('0' + (((n) / 1000)%10)),      \
00726   ('0' + (((n) / 100)%10)),       \
00727   ('0' + (((n) / 10)%10)),        \
00728   ('0' +  ((n) % 10))
00729
00730 /* Convert integer to hex digit literals.  */
00731 #define HEX(n)             \
00732   ('0' + ((n)>>28 & 0xF)), \
00733   ('0' + ((n)>>24 & 0xF)), \
00734   ('0' + ((n)>>20 & 0xF)), \
00735   ('0' + ((n)>>16 & 0xF)), \
00736   ('0' + ((n)>>12 & 0xF)), \
00737   ('0' + ((n)>>8  & 0xF)), \
00738   ('0' + ((n)>>4  & 0xF)), \
00739   ('0' + ((n)     & 0xF))
00740
00741 /* Construct a string literal encoding the version number. */
00742 #ifdef COMPILER_VERSION
00743 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "]";
00744
00745 /* Construct a string literal encoding the version number components. */
00746 #elif defined(COMPILER_VERSION_MAJOR)
00747 char const info_version[] = {
00748   'I', 'N', 'F', 'O', ':',
00749   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00750   COMPILER_VERSION_MAJOR,
00751 # ifdef COMPILER_VERSION_MINOR
00752   '.', COMPILER_VERSION_MINOR,
00753 #  ifdef COMPILER_VERSION_PATCH
00754   '.', COMPILER_VERSION_PATCH,
00755 #   ifdef COMPILER_VERSION_TWEAK
00756     '.', COMPILER_VERSION_TWEAK,
00757 #   endif
00758 #  endif
00759 # endif
00760   ']','\0'};
00761 #endif
00762
00763 /* Construct a string literal encoding the internal version number. */
00764 #ifdef COMPILER_VERSION_INTERNAL
00765 char const info_version_internal[] = {
00766   'I', 'N', 'F', 'O', ':',
00767   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00768   'i','n','t','e','r','n','a','l','[',
00769   COMPILER_VERSION_INTERNAL,']','\0'};
00770 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00771 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
      COMPILER_VERSION_INTERNAL_STR "]";
00772 #endif
00773
00774 /* Construct a string literal encoding the version number components. */
00775 #ifdef SIMULATE_VERSION_MAJOR
00776 char const info_simulate_version[] = {
00777   'I', 'N', 'F', 'O', ':',
00778   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00779   SIMULATE_VERSION_MAJOR,
00780 # ifdef SIMULATE_VERSION_MINOR
00781   '.', SIMULATE_VERSION_MINOR,
00782 #  ifdef SIMULATE_VERSION_PATCH
00783   '.', SIMULATE_VERSION_PATCH,
00784 #   ifdef SIMULATE_VERSION_TWEAK
00785     '.', SIMULATE_VERSION_TWEAK,
00786 #   endif
00787 #  endif
00788 # endif
00789   ']','\0'};
00790 #endif
00791
00792 /* Construct the string literal in pieces to prevent the source from
00793    getting matched.  Store it in a pointer rather than an array
00794    because some compilers will just produce instructions to fill the
00795    array rather than assigning a pointer to a static array.  */
00796 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00797 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00798
00799
00800
00801 #if !defined(__STDC__) && !defined(__clang__)
00802 # if defined(_MSC_VER) || defined(__ibmxl__) || defined(__IBMC__)
00803 #  define C_VERSION "90"
00804 # else
00805 #  define C_VERSION
00806 # endif
00807 #elif __STDC_VERSION__ > 201710L
00808 # define C_VERSION "23"
00809 #elif __STDC_VERSION__ >= 201710L
```

```
00810 # define C_VERSION "17"
00811 #elif __STDC_VERSION__ >= 201000L
00812 # define C_VERSION "11"
00813 #elif __STDC_VERSION__ >= 199901L
00814 # define C_VERSION "99"
00815 #else
00816 # define C_VERSION "90"
00817 #endif
00818 const char* info_language_standard_default =
00819   "INFO" ":" "standard_default[" C_VERSION "]";
00820
00821 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00822 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlC__) ||           \
00823      defined(__TI_COMPILER_VERSION__)) &&                                     \
00824   !defined(__STRICT_ANSI__)
00825   "ON"
00826 #else
00827   "OFF"
00828 #endif
00829 "]";
00830
00831 /*--------------------------------------------------------------------------*/
00832
00833 #ifdef ID_VOID_MAIN
00834 void main() {}
00835 #else
00836 # if defined(__CLASSIC_C__)
00837 int main(argc, argv) int argc; char *argv[];
00838 # else
00839 int main(int argc, char* argv[])
00840 # endif
00841 {
00842   int require = 0;
00843   require += info_compiler[argc];
00844   require += info_platform[argc];
00845   require += info_arch[argc];
00846 #ifdef COMPILER_VERSION_MAJOR
00847   require += info_version[argc];
00848 #endif
00849 #ifdef COMPILER_VERSION_INTERNAL
00850   require += info_version_internal[argc];
00851 #endif
00852 #ifdef SIMULATE_ID
00853   require += info_simulate[argc];
00854 #endif
00855 #ifdef SIMULATE_VERSION_MAJOR
00856   require += info_simulate_version[argc];
00857 #endif
00858 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00859   require += info_cray[argc];
00860 #endif
00861   require += info_language_standard_default[argc];
00862   require += info_language_extensions_default[argc];
00863   (void)argv;
00864   return require;
00865 }
00866 #endif
```

4.15 CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #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

4.15.1 Macro Definition Documentation

4.15.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

4.15.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 701 of file CMakeCXXCompilerId.cpp.

4.15.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 412 of file CMakeCXXCompilerId.cpp.

4.15.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 799 of file CMakeCXXCompilerId.cpp.

4.15.1.5 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 705 of file CMakeCXXCompilerId.cpp.

4.15.1.6 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 716 of file CMakeCXXCompilerId.cpp.

4.15.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 543 of file CMakeCXXCompilerId.cpp.

4.15.1.8 STRINGIFY

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

Definition at line 433 of file CMakeCXXCompilerId.cpp.

4.15.1.9 STRINGIFY_HELPER

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

Definition at line 432 of file CMakeCXXCompilerId.cpp.

4.15.2 Function Documentation

4.15.2.1 main()

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

Definition at line 830 of file CMakeCXXCompilerId.cpp.

4.15.3 Variable Documentation

4.15.3.1 info_arch

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

Definition at line 782 of file CMakeCXXCompilerId.cpp.

4.15.3.2 info_compiler

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

Definition at line 419 of file CMakeCXXCompilerId.cpp.

4.15.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

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



  "OFF"

"]"
```

Definition at line 818 of file CMakeCXXCompilerId.cpp.

4.15.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

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









  "98"

"]"
```

Definition at line 802 of file CMakeCXXCompilerId.cpp.

4.15.3.5 info_platform

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

Definition at line 781 of file CMakeCXXCompilerId.cpp.

4.16 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(__COMO__)
00019 # define COMPILER_ID "Comeau"
00020   /* __COMO_VERSION__ = VRR */
00021 # define COMPILER_VERSION_MAJOR DEC(__COMO_VERSION__ / 100)
00022 # define COMPILER_VERSION_MINOR DEC(__COMO_VERSION__ % 100)
00023
00024 #elif 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_CC)
00147 # define COMPILER_ID "SunPro"
00148 # if __SUNPRO_CC >= 0x5100
00149    /* __SUNPRO_CC = 0xVRRP */
00150 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»12)
00151 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xFF)
00152 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00153 # else
00154    /* __SUNPRO_CC = 0xVRP */
00155 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»8)
00156 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xF)
00157 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00158 # endif
00159
00160 #elif defined(__HP_aCC)
00161 # define COMPILER_ID "HP"
00162   /* __HP_aCC = VVRRPP */
00163 # define COMPILER_VERSION_MAJOR DEC(__HP_aCC/10000)
00164 # define COMPILER_VERSION_MINOR DEC(__HP_aCC/100 % 100)
00165 # define COMPILER_VERSION_PATCH DEC(__HP_aCC     % 100)
00166
00167 #elif defined(__DECCXX)
00168 # define COMPILER_ID "Compaq"
00169   /* __DECCXX_VER = VVRRTPPPP */
```

```
00170 # define COMPILER_VERSION_MAJOR DEC(__DECCXX_VER/10000000)
00171 # define COMPILER_VERSION_MINOR DEC(__DECCXX_VER/100000  % 100)
00172 # define COMPILER_VERSION_PATCH DEC(__DECCXX_VER         % 10000)
00173
00174 #elif defined(__IBMCPP__) && defined(__COMPILER_VER__)
00175 # define COMPILER_ID "zOS"
00176   /* __IBMCPP__ = VRP */
00177 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00178 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 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(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ >= 800
00198 # define COMPILER_ID "XL"
00199   /* __IBMCPP__ = VRP */
00200 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00201 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00202 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00203
00204 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ < 800
00205 # define COMPILER_ID "VisualAge"
00206   /* __IBMCPP__ = VRP */
00207 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00208 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00209 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 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(_CRAYC)
00228 # define COMPILER_ID "Cray"
00229 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00230 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00231
00232 #elif defined(__TI_COMPILER_VERSION__)
00233 # define COMPILER_ID "TI"
00234   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00235 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00236 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00237 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00238
00239 #elif defined(__CLANG_FUJITSU)
00240 # define COMPILER_ID "FujitsuClang"
00241 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00242 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00243 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00244 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00245
00246
00247 #elif defined(__FUJITSU)
00248 # define COMPILER_ID "Fujitsu"
00249 # if defined(__FCC_version__)
00250 #   define COMPILER_VERSION __FCC_version__
00251 # elif defined(__FCC_major__)
00252 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00253 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00254 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00255 # endif
00256 # if defined(__fcc_version)
```

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

```
00344 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00345 #  endif
00346 # endif
00347
00348 #elif defined(__GNUC__) || defined(__GNUG__)
00349 # define COMPILER_ID "GNU"
00350 # if defined(__GNUC__)
00351 #  define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00352 # else
00353 #  define COMPILER_VERSION_MAJOR DEC(__GNUG__)
00354 # endif
00355 # if defined(__GNUC_MINOR__)
00356 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00357 # endif
00358 # if defined(__GNUC_PATCHLEVEL__)
00359 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00360 # endif
00361
00362 #elif defined(_MSC_VER)
00363 # define COMPILER_ID "MSVC"
00364   /* _MSC_VER = VVRR */
00365 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00366 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00367 # if defined(_MSC_FULL_VER)
00368 #  if _MSC_VER >= 1400
00369     /* _MSC_FULL_VER = VVRRPPPPP */
00370 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00371 #  else
00372     /* _MSC_FULL_VER = VVRRPPPP */
00373 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00374 #  endif
00375 # endif
00376 # if defined(_MSC_BUILD)
00377 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00378 # endif
00379
00380 #elif defined(_ADI_COMPILER)
00381 # define COMPILER_ID "ADSP"
00382 #if defined(__VERSIONNUM__)
00383   /* __VERSIONNUM__ = 0xVVRRPPTT */
00384 #  define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ » 24 & 0xFF)
00385 #  define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ » 16 & 0xFF)
00386 #  define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ » 8 & 0xFF)
00387 #  define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00388 #endif
00389
00390 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00391 # define COMPILER_ID "IAR"
00392 # if defined(__VER__) && defined(__ICCARM__)
00393 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00394 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00395 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00396 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00397 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
      defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
      defined(__ICC8051__) || defined(__ICCSTM8__))
00398 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00399 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00400 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00401 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00402 # endif
00403
00404
00405 /* These compilers are either not known or too old to define an
00406   identification macro.  Try to identify the platform and guess that
00407   it is the native compiler.  */
00408 #elif defined(__hpux) || defined(__hpua)
00409 # define COMPILER_ID "HP"
00410
00411 #else /* unknown compiler */
00412 # define COMPILER_ID ""
00413 #endif
00414
00415 /* Construct the string literal in pieces to prevent the source from
00416    getting matched.  Store it in a pointer rather than an array
00417    because some compilers will just produce instructions to fill the
00418    array rather than assigning a pointer to a static array.  */
00419 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00420 #ifdef SIMULATE_ID
00421 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00422 #endif
00423
00424 #ifdef __QNXNTO__
00425 char const* qnxnto = "INFO" ":" "qnxnto[]";
00426 #endif
00427
00428 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
```

```
00429 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00430 #endif
00431
00432 #define STRINGIFY_HELPER(X) #X
00433 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00434
00435 /* Identify known platforms by name.  */
00436 #if defined(__linux) || defined(__linux__) || defined(linux)
00437 # define PLATFORM_ID "Linux"
00438
00439 #elif defined(__MSYS__)
00440 # define PLATFORM_ID "MSYS"
00441
00442 #elif defined(__CYGWIN__)
00443 # define PLATFORM_ID "Cygwin"
00444
00445 #elif defined(__MINGW32__)
00446 # define PLATFORM_ID "MinGW"
00447
00448 #elif defined(__APPLE__)
00449 # define PLATFORM_ID "Darwin"
00450
00451 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00452 # define PLATFORM_ID "Windows"
00453
00454 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00455 # define PLATFORM_ID "FreeBSD"
00456
00457 #elif defined(__NetBSD__) || defined(__NetBSD)
00458 # define PLATFORM_ID "NetBSD"
00459
00460 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00461 # define PLATFORM_ID "OpenBSD"
00462
00463 #elif defined(__sun) || defined(sun)
00464 # define PLATFORM_ID "SunOS"
00465
00466 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00467 # define PLATFORM_ID "AIX"
00468
00469 #elif defined(__hpux) || defined(__hpux__)
00470 # define PLATFORM_ID "HP-UX"
00471
00472 #elif defined(__HAIKU__)
00473 # define PLATFORM_ID "Haiku"
00474
00475 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00476 # define PLATFORM_ID "BeOS"
00477
00478 #elif defined(__QNX__) || defined(__QNXNTO__)
00479 # define PLATFORM_ID "QNX"
00480
00481 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00482 # define PLATFORM_ID "Tru64"
00483
00484 #elif defined(__riscos) || defined(__riscos__)
00485 # define PLATFORM_ID "RISCos"
00486
00487 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00488 # define PLATFORM_ID "SINIX"
00489
00490 #elif defined(__UNIX_SV__)
00491 # define PLATFORM_ID "UNIX_SV"
00492
00493 #elif defined(__bsdos__)
00494 # define PLATFORM_ID "BSDOS"
00495
00496 #elif defined(_MPRAS) || defined(MPRAS)
00497 # define PLATFORM_ID "MP-RAS"
00498
00499 #elif defined(__osf) || defined(__osf__)
00500 # define PLATFORM_ID "OSF1"
00501
00502 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00503 # define PLATFORM_ID "SCO_SV"
00504
00505 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00506 # define PLATFORM_ID "ULTRIX"
00507
00508 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00509 # define PLATFORM_ID "Xenix"
00510
00511 #elif defined(__WATCOMC__)
00512 # if defined(__LINUX__)
00513 #  define PLATFORM_ID "Linux"
00514
00515 # elif defined(__DOS__)
```

```
00516 #  define PLATFORM_ID "DOS"
00517
00518 # elif defined(__OS2__)
00519 #  define PLATFORM_ID "OS2"
00520
00521 # elif defined(__WINDOWS__)
00522 #  define PLATFORM_ID "Windows3x"
00523
00524 # elif defined(__VXWORKS__)
00525 #  define PLATFORM_ID "VxWorks"
00526
00527 # else /* unknown platform */
00528 #  define PLATFORM_ID
00529 # endif
00530
00531 #elif defined(__INTEGRITY)
00532 # if defined(INT_178B)
00533 #  define PLATFORM_ID "Integrity178"
00534
00535 # else /* regular Integrity */
00536 #  define PLATFORM_ID "Integrity"
00537 # endif
00538
00539 # elif defined(_ADI_COMPILER)
00540 #  define PLATFORM_ID "ADSP"
00541
00542 #else /* unknown platform */
00543 # define PLATFORM_ID
00544
00545 #endif
00546
00547 /* For windows compilers MSVC and Intel we can determine
00548    the architecture of the compiler being used.  This is because
00549    the compilers do not have flags that can change the architecture,
00550    but rather depend on which compiler is being used
00551 */
00552 #if defined(_WIN32) && defined(_MSC_VER)
00553 # if defined(_M_IA64)
00554 #  define ARCHITECTURE_ID "IA64"
00555
00556 # elif defined(_M_ARM64EC)
00557 #  define ARCHITECTURE_ID "ARM64EC"
00558
00559 # elif defined(_M_X64) || defined(_M_AMD64)
00560 #  define ARCHITECTURE_ID "x64"
00561
00562 # elif defined(_M_IX86)
00563 #  define ARCHITECTURE_ID "X86"
00564
00565 # elif defined(_M_ARM64)
00566 #  define ARCHITECTURE_ID "ARM64"
00567
00568 # elif defined(_M_ARM)
00569 #  if _M_ARM == 4
00570 #   define ARCHITECTURE_ID "ARMV4I"
00571 #  elif _M_ARM == 5
00572 #   define ARCHITECTURE_ID "ARMV5I"
00573 #  else
00574 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00575 #  endif
00576
00577 # elif defined(_M_MIPS)
00578 #  define ARCHITECTURE_ID "MIPS"
00579
00580 # elif defined(_M_SH)
00581 #  define ARCHITECTURE_ID "SHx"
00582
00583 # else /* unknown architecture */
00584 #  define ARCHITECTURE_ID ""
00585 # endif
00586
00587 #elif defined(__WATCOMC__)
00588 # if defined(_M_I86)
00589 #  define ARCHITECTURE_ID "I86"
00590
00591 # elif defined(_M_IX86)
00592 #  define ARCHITECTURE_ID "X86"
00593
00594 # else /* unknown architecture */
00595 #  define ARCHITECTURE_ID ""
00596 # endif
00597
00598 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00599 # if defined(__ICCARM__)
00600 #  define ARCHITECTURE_ID "ARM"
00601
00602 # elif defined(__ICCRX__)
```

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

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

```
00776
00777 /* Construct the string literal in pieces to prevent the source from
00778    getting matched.  Store it in a pointer rather than an array
00779    because some compilers will just produce instructions to fill the
00780    array rather than assigning a pointer to a static array.  */
00781 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00782 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00783
00784
00785
00786 #if defined(__INTEL_COMPILER) && defined(_MSVC_LANG) && _MSVC_LANG < 201403L
00787 #  if defined(__INTEL_CXX11_MODE__)
00788 #    if defined(__cpp_aggregate_nsdmi)
00789 #      define CXX_STD 201402L
00790 #    else
00791 #      define CXX_STD 201103L
00792 #    endif
00793 #  else
00794 #    define CXX_STD 199711L
00795 #  endif
00796 #elif defined(_MSC_VER) && defined(_MSVC_LANG)
00797 #  define CXX_STD _MSVC_LANG
00798 #else
00799 #  define CXX_STD __cplusplus
00800 #endif
00801
00802 const char* info_language_standard_default = "INFO" ":" "standard_default["
00803 #if CXX_STD > 202002L
00804   "23"
00805 #elif CXX_STD > 201703L
00806   "20"
00807 #elif CXX_STD >= 201703L
00808   "17"
00809 #elif CXX_STD >= 201402L
00810   "14"
00811 #elif CXX_STD >= 201103L
00812   "11"
00813 #else
00814   "98"
00815 #endif
00816 "]";
00817
00818 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00819 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlC__) ||           \
00820      defined(__TI_COMPILER_VERSION__)) &&                                     \
00821   !defined(__STRICT_ANSI__)
00822   "ON"
00823 #else
00824   "OFF"
00825 #endif
00826 "]";
00827
00828 /*--------------------------------------------------------------------------*/
00829
00830 int main(int argc, char* argv[])
00831 {
00832   int require = 0;
00833   require += info_compiler[argc];
00834   require += info_platform[argc];
00835   require += info_arch[argc];
00836 #ifdef COMPILER_VERSION_MAJOR
00837   require += info_version[argc];
00838 #endif
00839 #ifdef COMPILER_VERSION_INTERNAL
00840   require += info_version_internal[argc];
00841 #endif
00842 #ifdef SIMULATE_ID
00843   require += info_simulate[argc];
00844 #endif
00845 #ifdef SIMULATE_VERSION_MAJOR
00846   require += info_simulate_version[argc];
00847 #endif
00848 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00849   require += info_cray[argc];
00850 #endif
00851   require += info_language_standard_default[argc];
00852   require += info_language_extensions_default[argc];
00853   (void)argv;
00854   return require;
00855 }
```

Index