My Project

Generated by Doxygen 1.8.6

Thu Jan 21 2016 13:43:28

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

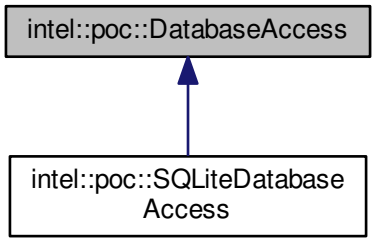
Class Documentation

3.1 intel::poc::DatabaseAccess Class Reference

Database abstract base class.

`#include <databaseaccess.h>`

Inheritance diagram for intel::poc::DatabaseAccess:



Public Member Functions

- virtual ~DatabaseAccess ()
- virtual bool init (const std::string &database_path, const Json::Value &data_schema, bool clean)=0
- virtual bool putData (const Json::Value &data_values)=0
- virtual Json::Value getData (const Json::Value &params)=0

Protected Member Functions

- DatabaseAccess ()

  constructor

3.1.1 Detailed Description

Database abstract base class.

DatabaseAccess is an abstract class that represents a TSDV database that contains the raw data.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 virtual intel::poc::DatabaseAccess::~DatabaseAccess ( ) [inline], [virtual]

A destructor

3.1.3 Member Function Documentation

3.1.3.1 virtual Json::Value intel::poc::DatabaseAccess::getData (const Json::Value & *params*) [pure virtual]

Retrieve data points requested from the specified time time range and granularity

Parameters

in	*params*	A Json::Value object that specifies the search query parameters, currently supported parameters are: startDate: the start date of the data points endDate: the end date of the data points numOfPoints: the maximum number of of points that is a downsampled average of original data points. metrics: (future, not yet supported)

Queries constructed in this form:

{ "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "metrics" : ["calories", "gsr", "heart_rate", "body_temp", "steps"]} }

Returns

A Json::Value object that represent the array of data point elements in this format: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] } Note: In the event of an error, the function will return an empty response in this format: { "startDate" : "", "endDate" : "", "points" : [] }

Implemented in intel::poc::SQLiteDatabaseAccess.

3.1.3.2 virtual bool intel::poc::DatabaseAccess::init (const std::string & *database_path,* const Json::Value & *data_schema,* bool *clean*) [pure virtual]

Initialize DatabaseAccess. Must be called exactly once before using the database class.

Parameters

in	*dataSchema*	Json::Value that represents the table name and the mapping of column names to their data types. Used for creating or reading the databases. For example: { "table": "data", "date_key_column": "date", "columns": { "date": "TEXT", "calories": "INT", "steps": "INT", "body_temp": "REAL" } } Note: The column identified by the "date_key_column" field will be treated as the primary key. It MUST be present in the column list and MUST be of type "TEXT". Dates in database should be of the format: "YYYY-MM-DD HH:MMZ".

in	*clean*	Indicate if backend should initialize with clean database, if it is set to true, all existing data will be deleted.

Return values

true	Succesfully initialized
false	Failed to initialize

Implemented in intel::poc::SQLiteDatabaseAccess.

3.1.3.3 virtual bool intel::poc::DatabaseAccess::putData (const Json::Value & *data_values*) [pure virtual]

Adds new data points to the database

Parameters

in	*data_values*	A Json::Value object that represents data points in this form: { "startDate" -: "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate"-:0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] }

Please note that fields must match the columns defined in the data_schema parameter passed to init.

Return values

true	Succesfully added data to library
false	Failed to add data to library

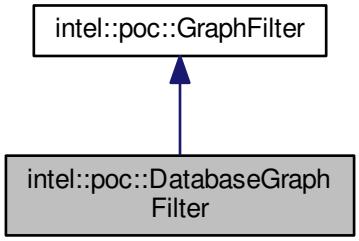
Implemented in intel::poc::SQLiteDatabaseAccess.

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

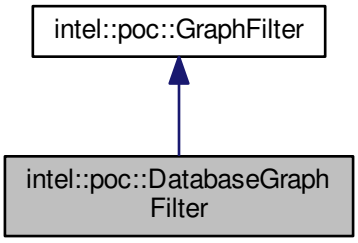
- include/graphfilter/databaseaccess.h

3.2 intel::poc::DatabaseGraphFilter Class Reference

Inheritance diagram for intel::poc::DatabaseGraphFilter:



Collaboration diagram for intel::poc::DatabaseGraphFilter:



Public Member Functions

- DatabaseGraphFilter ()

  constructor
- ~DatabaseGraphFilter ()

  destructor
- const std::string id () const

  API.
- bool init (const std::string &cache_setup, const std::string &dataSchema, const std::string &database_path, bool clean)
- bool addData (const std::string &data_values)
- std::string getData (const std::string &params)

Additional Inherited Members

3.2.1 Member Function Documentation

3.2.1.1 bool intel::poc::DatabaseGraphFilter::addData (const std::string & *data_values*) [virtual]

Adds new data points to the database

Parameters

in	*data_values*	A JSON formatted string that represents data points in this form: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] }

Please note that fields must match the columns defined in the data_schema parameter passed to init.

Return values

true	Succesfully added data to library

false	Failed to add data to library

Implements intel::poc::GraphFilter.

3.2.1.2 std::string intel::poc::DatabaseGraphFilter::getData (const std::string & *params*) [virtual]

Retrieve data points requested from the specified time time range and granularity

Parameters

in	*params*	A JSON formatted string that specifies the searh query parameters, currently supported parameters are: startDate: the start date of the data points endDate: the end date of the data points numOfPoints: the maximum number of of points that is an avereage of the downsampled data points. metrics: (future, not yet supported)

Queries constructed in this form:

{ "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "metrics" : ["calories", "gsr", "heart_rate", "body_temp", "steps"], "numOfPoints" : 100 } }

Returns

A JSON formatted string that represent the array of data point elements in this format: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] } Note: In the event of an error, the function will return an empty response in this format: { "startDate" : "", "endDate" : "", "points" : [] }

Implements intel::poc::GraphFilter.

3.2.1.3 bool intel::poc::DatabaseGraphFilter::init (const std::string & *cache_setup,* const std::string & *data_schema,* const std::string & *database_path,* bool *clean*) [virtual]

Initialize GraphFilter library, must be called exactly once before using the library.

Parameters

in	*cache_setup*	JSON string that describes how the cache should be set up, including how the cache should pre-calculate and store downsampled versions of the data. If this parameter is empty, it will be equivalent to passing false for "useCache" and the cache will not be used. For example: { "useCache": true, "downsamplingFilter": "TIME_WEIGHTED_POINTS", "cacheRawData": true, "fetchAhead": 1, "fetchBehind": 2, "downsamplingLevels": [{ "duration": 31536000, "numOfPoints": 100 }, { "duration": 86400, "numOfPoints": 100 },] } Note: In the above example, the cache will pre-calculate and store two levels of downsampled data: in the first, the raw data will be downsampled to 100 points for every 31536000 seconds (1 year); in the second, the raw data will be downsampled to 100 points for every day. Note: If not present, "cacheRawData" will be treated as false and only downsampled data will be cached. Note: "fetchAhead" and "fetchBehind" indicate the number of multiples of the current cache putData request's duration should be pre-fetched and cached. For example, the above values would mean that if a cache putData call was made for a time period spanning one day, the two previous days and one following day (for a total of 4 days, including the original request) would be fetched, downsampled, and stored in the cache. Note: If not present, "downsamplingFilter" will default to DataFilter::TIME_WEIGHTED_POINTS.
in	*data_schema*	JSON string that represents the mapping of column names to their data types. Used for creating the databases or data structures used to implement the data cache. For example: { "table": "data", "date_key_column": "date", "columns": { "date": "TEXT", "calories": "INT", "steps": "INT", "body_temp": "REAL" } } Note: The column identified by the "date_key_column" field will be treated as the primary key. It MUST be present in the column list and MUST be of type "TEXT". Dates in database should be of the format: "YYYY-MM-DD HH:MMZ".
in	*database_path*	Database backend path, e.g. "`/path/to/database.db`"
in	*clean*	Indicate if backend should initialize with clean database, if it is set to true, all existing data will be deleted.

Return values

true	Succesfully initialized
false	Failed to initialize

Implements intel::poc::GraphFilter.

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

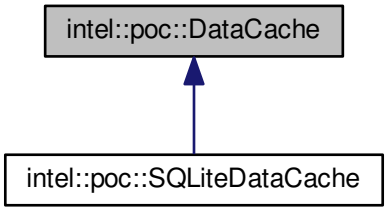
- include/graphfilter/databasegraphfilter.h
- src/databasegraphfilter.cpp

3.3 intel::poc::DataCache Class Reference

Abstract data cache base class.

`#include <datacache.h>`

Inheritance diagram for intel::poc::DataCache:



Public Member Functions

- virtual ~DataCache ()
- virtual bool init (const Json::Value &cache_setup, const Json::Value &data_schema, bool clean)=0
- virtual void cacheData (const std::string &start_date, const std::string &end_date)=0
- virtual Json::Value getData (const Json::Value &params)=0

Protected Member Functions

- DataCache ()

  constructor

Protected Attributes

- bool **initialized_**

3.3.1 Detailed Description

Abstract data cache base class.

DataCache is an abstract class that represents a faster, probably in-memory cache for already-accessed and pre-calculated data.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 virtual intel::poc::DataCache::~DataCache ( ) [inline], [virtual]

A destructor

3.3.3 Member Function Documentation

3.3.3.1 virtual void intel::poc::DataCache::cacheData (const std::string & *start_date*, const std::string & *end_date*) [pure virtual]

Adds new data points to the cache. Because processing is done done asynchronously on a separate thread, there is no return value or indication of success of the actual database put.

Parameters

in	*start_date*	A date-time string indicating the start of the time interval for which data should be retreived and cached. Should be of the format: "YYYY-MM-DD HH:MMZ".
in	*start_date*	A date-time string indicating the end of the time interval for which data should be retreived and cached. Should be of the format: "YYYY-MM-DD HH:MMZ".

Implemented in intel::poc::SQLiteDataCache.

3.3.3.2 virtual Json::Value intel::poc::DataCache::getData (const Json::Value & *params*) [pure virtual]

Retrieve data points requested from the specified time time range and granularity

Parameters

in	*params*	A Json::Value object that specifies the search query parameters, currently supported parameters are: startDate: the start date of the data points endDate: the end date of the data points numOfPoints: the maximum number of of points that is a downsampled average of original data points. metrics: (future, not yet supported)

Queries constructed in this form:

{ "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "metrics" : ["calories", "gsr", "heart_rate", "body_temp", "steps"], "numOfPoints" : 100 } }

Returns

A Json::Value object that represent the array of data point elements in this format: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.-27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] } Note: In the event of an error, OR in the event the cache does not contain the data requested, the function will return an empty response in this format: { "startDate" : "", "endDate" : "", "points" : [] }

Implemented in intel::poc::SQLiteDataCache.

3.3.3.3 virtual bool intel::poc::DataCache::init (const Json::Value & *cache_setup,* const Json::Value & *data_schema,* bool *clean*) [pure virtual]

Initialize DataCache. Must be called exactly once before using the library.

Parameters

in	*cache_setup*	Json::Value object that describes how the cache should be set up, including how the cache should pre-calculate and store downsampled versions of the data. Required parameter. For example: { "cacheRawData": true, "downsamplingFilter": "TIME_WEIGHTED_POINTS", "fetchAhead": 1, "fetchBehind": 2, "downsamplingLevels": [{ "duration": 31536000, "numOfPoints": 100 }, { "duration": 86400, "numOfPoints": 100 },] } Note: In the above example, the cache will pre-calculate and store two levels of downsampled data: in the first, the raw data will be downsampled to 100 points for every 31536000 seconds (1 year); in the second, the raw data will be downsampled to 100 points for every day. Note: If not present, "cacheRawData" will be treated as false and only downsampled data will be cached. Note: "fetchAhead" and "fetchBehind" indicate the number of multiples of the current putData request's duration should be pre-fetched and cached. For example, the above values would mean that if a putData request came in for a time period spanning one day, the two previous days and one following day (for a total of 4 days, including the original request) would be fetched, downsampled, and stored in the cache. Note: If not present, "downsamplingFilter" will default to DataFilter::TIME_WEIGHTED_POINTS.
in	*data_schema*	Json::Value object that represents the mapping of column names to their data types. Used for creating the databases or data structures used to implement the data cache. For example: { "table": "data", "date_key_column": "date", "columns": { "date": "TEXT", "calories": "INT", "steps": "INT", "body_temp": "REAL" } } Note: The column identified by the "date_key_column" field will be treated as the primary key. It MUST be present in the column list and MUST be of type "TEXT". Dates in database should be of the format: "YYYY-MM-DD HH:MMZ".
in	*clean*	Indicate if backend should initialize with clean database, if it is set to true, all existing cached data will be deleted.

Return values

true	Succesfully initialized
false	Failed to initialize

Implemented in intel::poc::SQLiteDataCache.

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

- include/graphfilter/datacache.h

3.4 intel::poc::DataFilter Class Reference

Public Types

- enum FilterType { **POINTS**, **TIME_WEIGHTED_POINTS**, **TIME_WEIGHTED_TIME** }

Static Public Member Functions

- static bool init (const std::string &date_key)
- static Json::Value applyFilter (const Json::Value &data, const std::map< std::string, std::string > &data_schema, int num_of_points, FilterType filter)
- static FilterType getType (std::string filter_string)

3.4.1 Member Enumeration Documentation

3.4.1.1 enum intel::poc::DataFilter::FilterType [strong]

Enum representing the valid filter types

3.4.2 Member Function Documentation

3.4.2.1 Json::Value intel::poc::DataFilter::applyFilter (const Json::Value & *data*, const std::map< std::string, std::string > & *data_schema*, int *num_of_points*, FilterType *filter*) [static]

Downsample the given data using the given filter to the given number of points.

Parameters

in	*data*	A Json::Value object that represent the array of data point elements in this format: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.-1 "steps", 0 }] }
in	*data_schema*	Json::Value object that represents the mapping of column names to their data types. Used to parse data elements correctly. For example: { "table": "data", "date_key_column": "date", "columns": { "date": "TEXT", "calories": "INT", "steps": "INT", "body_temp": "REAL" } }
in	*num_of_points*	The maximum number of points you wish the data downsampled to. Note that you are not guaranteed to receive this number of points.

Returns

a Json::Value representing the downsampled data

3.4.2.2 DataFilter::FilterType intel::poc::DataFilter::getType (std::string *filter_string*) [static]

Helper function to get an enum type given its equivalent string.

Parameters

in	*filter_string*	String representing a filter type. Should be textually equivalent to one of the valid filter types.

Returns

The corresponding FilterType value.

3.4.2.3 bool intel::poc::DataFilter::init (const std::string & *date_key*) [static]

Initialize DataFilter. Must be called exactly once before using the library.

Parameters

in	*date_key_column*	String identifying the field in your data points that represents the date timestamp field

Return values

true	Succesfully initialized
false	Failed to initialize

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

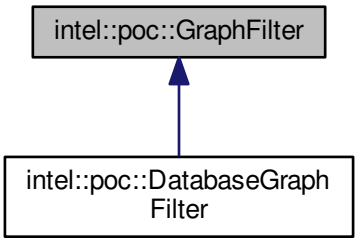
- include/graphfilter/datafilter.h
- src/datafilter.cpp

3.5 intel::poc::GraphFilter Class Reference

GraphFilter prefetch libary.

`#include <graphfilter.h>`

Inheritance diagram for intel::poc::GraphFilter:



Public Member Functions

- virtual ~GraphFilter ()
- virtual const std::string id () const =0
- virtual bool init (const std::string &cache_setup, const std::string &data_schema, const std::string &database_path, bool clean)=0
- virtual bool addData (const std::string &data_values)=0
- virtual std::string getData (const std::string &params)=0

Static Public Member Functions

- static GraphFilter & instance ()

Protected Member Functions

- GraphFilter ()

  constructor

Protected Attributes

- bool **initialized_**

3.5.1 Detailed Description

GraphFilter prefetch libary.

GraphFilter is an abstract class that represents the pre-fetch data library that caches time-series data fetches from an application data backend. It's a singleton that can be accessed by using the static instance() method.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 virtual intel::poc::GraphFilter::~GraphFilter ( ) [inline], [virtual]

A destructor

3.5.3 Member Function Documentation

3.5.3.1 virtual bool intel::poc::GraphFilter::addData (const std::string & *data_values*) [pure virtual]

Adds new data points to the database

Parameters

in	*data_values*	A JSON formatted string that represents data points in this form: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] }

Please note that fields must match the columns defined in the data_schema parameter passed to init.

Return values

true	Succesfully added data to library
false	Failed to add data to library

Implemented in intel::poc::DatabaseGraphFilter.

3.5.3.2 virtual std::string intel::poc::GraphFilter::getData (const std::string & *params*) [pure virtual]

Retrieve data points requested from the specified time time range and granularity

Parameters

in	*params*	A JSON formatted string that specifies the searh query parameters, currently supported parameters are: startDate: the start date of the data points endDate: the end date of the data points numOfPoints: the maximum number of of points that is an avereage of the downsampled data points. metrics: (future, not yet supported)

Queries constructed in this form:

{ "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "metrics" : ["calories", "gsr", "heart_rate", "body_temp", "steps"], "numOfPoints" : 100 } }

Returns

A JSON formatted string that represent the array of data point elements in this format: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] } Note: In the event of an error, the function will return an empty response in this format: { "startDate" : "", "endDate" : "", "points" : [] }

Implemented in intel::poc::DatabaseGraphFilter.

3.5.3.3 virtual const std::string intel::poc::GraphFilter::id ( ) const [pure virtual]

Back end database identifier

Returns

Unique id identifying the backend database.

Implemented in intel::poc::DatabaseGraphFilter.

3.5.3.4 virtual bool intel::poc::GraphFilter::init (const std::string & *cache_setup,* const std::string & *data_schema,* const std::string & *database_path,* bool *clean*) [pure virtual]

Initialize GraphFilter library, must be called exactly once before using the library.

Parameters

in	*cache_setup*	JSON string that describes how the cache should be set up, including how the cache should pre-calculate and store downsampled versions of the data. If this parameter is empty, it will be equivalent to passing false for "useCache" and the cache will not be used. For example: { "useCache": true, "downsamplingFilter": "TIME_WEIGHTED_POINTS", "cacheRawData": true, "fetchAhead": 1, "fetchBehind": 2, "downsamplingLevels": [{ "duration": 31536000, "numOfPoints": 100 }, { "duration": 86400, "numOfPoints": 100 },] } Note: In the above example, the cache will pre-calculate and store two levels of downsampled data: in the first, the raw data will be downsampled to 100 points for every 31536000 seconds (1 year); in the second, the raw data will be downsampled to 100 points for every day. Note: If not present, "cacheRawData" will be treated as false and only downsampled data will be cached. Note: "fetchAhead" and "fetchBehind" indicate the number of multiples of the current cache putData request's duration should be pre-fetched and cached. For example, the above values would mean that if a cache putData call was made for a time period spanning one day, the two previous days and one following day (for a total of 4 days, including the original request) would be fetched, downsampled, and stored in the cache. Note: If not present, "downsamplingFilter" will default to DataFilter::TIME_WEIGHTED_POINTS.

in	*data_schema*	JSON string that represents the mapping of column names to their data types. Used for creating the databases or data structures used to implement the data cache. For example: { "table": "data", "date_key_column": "date", "columns": { "date": "TEXT", "calories": "INT", "steps": "INT", "body_temp": "REAL" } } Note: The column identified by the "date_key_column" field will be treated as the primary key. It MUST be present in the column list and MUST be of type "TEXT". Dates in database should be of the format: "YYYY-MM-DD HH:MMZ".
in	*database_path*	Database backend path, e.g. "`/path/to/database.db`"
in	*clean*	Indicate if backend should initialize with clean database, if it is set to true, all existing data will be deleted.

Return values

true	Succesfully initialized
false	Failed to initialize

Implemented in intel::poc::DatabaseGraphFilter.

3.5.3.5 GraphFilter & intel::poc::GraphFilter::instance ( ) [static]

An instance method This method retrieves the singleton object

Returns

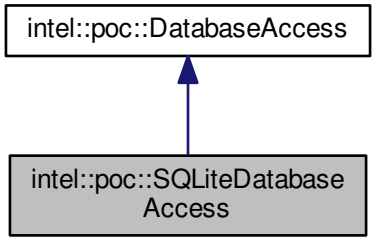
GraphFilter object

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

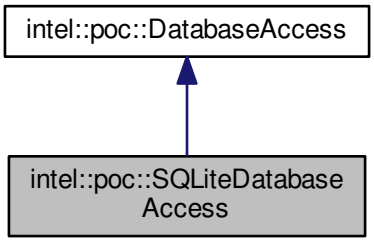
- include/graphfilter/graphfilter.h
- src/graphfilter.cpp

3.6 intel::poc::SQLiteDatabaseAccess Class Reference

Inheritance diagram for intel::poc::SQLiteDatabaseAccess:



Collaboration diagram for intel::poc::SQLiteDatabaseAccess:



Public Member Functions

- bool init (const std::string &database_path, const Json::Value &data_schema, bool clean)
- bool putData (const Json::Value &data_values)
- Json::Value getData (const Json::Value &params)

Static Public Member Functions

- static DatabaseAccess & **instance** ()

Protected Member Functions

- SQLiteDatabaseAccess ()
  constructor
- bool checkDatabase ()
  private API
- bool **openDatabase** (const std::string &database_path)
- void **createDatabase** ()
- void **executeQuery** (const std::string &sql_query)

Protected Attributes

- sqlite3 ∗ **database_**
- std::string **table_name_**
- std::map< std::string, std::string > **data_schema_**
- std::string **date_key_column_**
- bool **initialized_**

3.6.1 Member Function Documentation

3.6.1.1 Json::Value intel::poc::SQLiteDatabaseAccess::getData (const Json::Value & *params*) [virtual]

Retrieve data points requested from the specified time time range and granularity

Parameters

in	*params*	A Json::Value object that specifies the search query parameters, currently supported parameters are: startDate: the start date of the data points endDate: the end date of the data points numOfPoints: the maximum number of of points that is a downsampled average of original data points. metrics: (future, not yet supported)

Queries constructed in this form:

{ "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "metrics" : ["calories", "gsr", "heart_rate", "body_temp", "steps"]} }

Returns

A Json::Value object that represent the array of data point elements in this format: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] } Note: In the event of an error, the function will return an empty response in this format: { "startDate" : "", "endDate" : "", "points" : [] }

Implements intel::poc::DatabaseAccess.

3.6.1.2 bool intel::poc::SQLiteDatabaseAccess::init (const std::string & *database_path,* const Json::Value & *data_schema,* bool *clean*) [virtual]

Initialize DatabaseAccess. Must be called exactly once before using the database class.

Parameters

in	*dataSchema*	Json::Value that represents the table name and the mapping of column names to their data types. Used for creating or reading the databases. For example: { "table": "data", "date_key_column": "date", "columns": { "date": "TEXT", "calories": "INT", "steps": "INT", "body_temp": "REAL" } } Note: The column identified by the "date_key_column" field will be treated as the primary key. It MUST be present in the column list and MUST be of type "TEXT". Dates in database should be of the format: "YYYY-MM-DD HH:MMZ".
in	*clean*	Indicate if backend should initialize with clean database, if it is set to true, all existing data will be deleted.

Return values

true	Succesfully initialized
false	Failed to initialize

Implements intel::poc::DatabaseAccess.

3.6.1.3 bool intel::poc::SQLiteDatabaseAccess::putData (const Json::Value & *data_values*) [virtual]

Adds new data points to the database

Parameters

in	*data_values*	A Json::Value object that represents data points in this form: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] }

Please note that fields must match the columns defined in the data_schema parameter passed to init.

Return values

true	Succesfully added data to library
false	Failed to add data to library

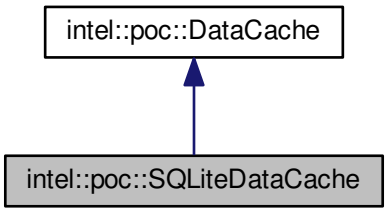
Implements intel::poc::DatabaseAccess.

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

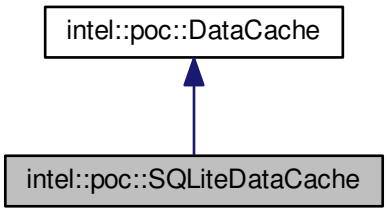
- include/graphfilter/sqlitedatabaseaccess.h
- src/sqlitedatabaseaccess.cpp

3.7 intel::poc::SQLiteDataCache Class Reference

Inheritance diagram for intel::poc::SQLiteDataCache:



Collaboration diagram for intel::poc::SQLiteDataCache:



Public Member Functions

- bool init (const Json::Value &cache_setup, const Json::Value &data_schema, bool clean)
- void cacheData (const std::string &start_date, const std::string &end_date)
- Json::Value getData (const Json::Value &params)

Static Public Member Functions

- static DataCache & **instance** ()

Protected Member Functions

- SQLiteDataCache ()

  constructor
- void cacheDataAsync (const std::string &start_date, const std::string &end_date)

  private API
- bool **getAndPutData** (const std::string &start_date, const std::string &end_date)
- bool **downsampleAndPutData** (int level, const Json::Value &data_values)
- bool **putDataTable** (const std::string &table_name, const Json::Value &points)
- long **timeStringToEpochSeconds** (const std::string &time_string)
- std::string **updateTimeString** (const std::string &time_string, long offset)
- long **getDurationNumPoints** (const std::string &start_date, const std::string &end_date, int level)
- bool **clearDatabaseRange** (const std::string &table_name, const std::string &start_date, const std::string &end_date)
- std::string cacheContains (const std::string &startDate, const std::string &endDate, int num_of_points)

  private API
- std::map< std::string, std::string > getCacheDifference (std::string start_date, std::string end_date)
- bool **openDatabase** ()
- void **createDatabase** ()
- void **executeQuery** (const std::string &sql_query)
- std::vector< std::string > **executeSelectQuery** (const std::string &sql_query, int num_of_col)

Protected Attributes

- sqlite3 * **database_**
- std::string **table_name_**
- std::map< std::string, std::string > **data_schema_**
- bool **cache_raw_data_**
- int **fetch_ahead_**
- int **fetch_behind_**
- DataFilter::FilterType **downsampling_filter_**
- std::vector< std::map < std::string, long > > **cache_levels_**
- std::string **date_key_column_**
- std::map< std::string, std::string > **cache_data_bounds_**
- std::mutex **put_data_mutex_**
- std::mutex **cache_data_bounds_mutex_**
- bool **initialized_**

Static Protected Attributes

- static const std::string **database_path_** = ":memory:"

3.7.1 Member Function Documentation

3.7.1.1 std::string intel::poc::SQLiteDataCache::cacheContains (const std::string & *start_date*, const std::string & *end_date*, int *num_of_points*) [protected]

private API

Checks whether or not the cache contains data for the given dates and for the given number of points.

Parameters

in	*start_date*	timestampof the format: "YYYY-MM-DD HH:MMZ"
in	*end_date*	timestampof the format: "YYYY-MM-DD HH:MMZ"
in	*num_of_points*	the number of points requested

Return values

<*table_name*>	The name of the table that will satisfy the request
""	Failed to find a cache level that will satisfy the request

3.7.1.2 void intel::poc::SQLiteDataCache::cacheData (const std::string & *start_date*, const std::string & *end_date*) [virtual]

Adds new data points to the cache. Because processing is done done asynchronously on a separate thread, there is no return value or indication of success of the actual database put.

Parameters

in	*start_date*	A date-time string indicating the start of the time interval for which data should be retreived and cached. Should be of the format: "YYYY-MM-DD HH:MMZ".
in	*start_date*	A date-time string indicating the end of the time interval for which data should be retreived and cached. Should be of the format: "YYYY-MM-DD HH:MMZ".

Implements intel::poc::DataCache.

3.7.1.3 std::map< std::string, std::string > intel::poc::SQLiteDataCache::getCacheDifference (std::string *start_date*, std::string *end_date*) [protected]

Returns the difference between your search interval and the cache. That is, returns a map of <start_date,end_date> pairs representing the intervals in your search range not already present in the cache.

Note that this function relies on the guaranteed sorted ordering of keys in std::map.

Parameters

in	*start_date*	timestampof the format: "YYYY-MM-DD HH:MMZ"
in	*end_date*	timestampof the format: "YYYY-MM-DD HH:MMZ"

Returns

The map containing the difference between your search interval and the cache. If an empty map is returned, the cache already contains your entire search interval.

3.7.1.4 Json::Value intel::poc::SQLiteDataCache::getData (const Json::Value & *params*) [virtual]

Retrieve data points requested from the specified time time range and granularity

Parameters

in	*params*	A Json::Value object that specifies the search query parameters, currently supported parameters are: startDate: the start date of the data points endDate: the end date of the data points numOfPoints: the maximum number of of points that is a downsampled average of original data points. metrics: (future, not yet supported)

Queries constructed in this form:

{ "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "metrics" : ["calories", "gsr", "heart_rate", "body_temp", "steps"], "numOfPoints" : 100 } }

Returns

A Json::Value object that represent the array of data point elements in this format: { "startDate" : "2015-03-03 00:00Z", "endDate" : "2015-03-03 23:59Z", "points" : [{ "date":"2015-03-03 00:00Z", "calories":2.0, "gsr":"4.-27263e-05", "heart_rate":0 "body_temp":82.4 "steps", 0 }, ... { "date":"2015-03-03 23:59Z", "calories":1.0, "gsr":"4.22559e-05", "heart_rate":68 "body_temp":85.1 "steps", 0 }] } Note: In the event of an error, OR in the event the cache does not contain the data requested, the function will return an empty response in this format: { "startDate" : "", "endDate" : "", "points" : [] }

Implements intel::poc::DataCache.

3.7.1.5 bool intel::poc::SQLiteDataCache::init (const Json::Value & *cache_setup,* const Json::Value & *data_schema,* bool *clean*) [virtual]

Initialize DataCache. Must be called exactly once before using the library.

Parameters

in	*cache_setup*	Json::Value object that describes how the cache should be set up, including how the cache should pre-calculate and store downsampled versions of the data. Required parameter. For example: { "cacheRawData": true, "downsamplingFilter": "TIME_WEIGHTED_POINTS", "fetchAhead": 1, "fetchBehind": 2, "downsamplingLevels": [{ "duration": 31536000, "numOfPoints": 100 }, { "duration": 86400, "numOfPoints": 100 },] } Note: In the above example, the cache will pre-calculate and store two levels of downsampled data: in the first, the raw data will be downsampled to 100 points for every 31536000 seconds (1 year); in the second, the raw data will be downsampled to 100 points for every day. Note: If not present, "cacheRawData" will be treated as false and only downsampled data will be cached. Note: "fetchAhead" and "fetchBehind" indicate the number of multiples of the current putData request's duration should be pre-fetched and cached. For example, the above values would mean that if a putData request came in for a time period spanning one day, the two previous days and one following day (for a total of 4 days, including the original request) would be fetched, downsampled, and stored in the cache. Note: If not present, "downsamplingFilter" will default to DataFilter::TIME_WEIGHTED_POINTS.

in	*data_schema*	Json::Value object that represents the mapping of column names to their data types. Used for creating the databases or data structures used to implement the data cache. For example: { "table": "data", "date_key_column": "date", "columns": { "date": "TEXT", "calories": "INT", "steps": "INT", "body_temp": "REAL" } } Note: The column identified by the "date_key_column" field will be treated as the primary key. It MUST be present in the column list and MUST be of type "TEXT". Dates in database should be of the format: "YYYY-MM-DD HH:MMZ".
in	*clean*	Indicate if backend should initialize with clean database, if it is set to true, all existing cached data will be deleted.

Return values

true	Succesfully initialized
false	Failed to initialize

Implements intel::poc::DataCache.

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

- include/graphfilter/sqlitedatacache.h
- src/sqlitedatacache.cpp

Index