Institutt for energiteknikk
OECD Halden Reactor Project

# The Software Bus Historic Log

# REFERENCE MANUAL

**SOFTWARE BUS DOCUMENTATION**

Revision: 4.6

# Institutt for energiteknikk
# OECD Halden Reactor Project

The Software Bus development team can be contacted at the following e-mail address:
**softbus@hrp.no**
For further information, see the Software Bus WWW pages at URL:
**http://www.ife.no/swbus**



Published by : Institutt for Energiteknikk, OECD Halden Reactor Project

Date : June 07

Revision : R4.6

# Table Of Contents

# Introduction

A historic log module has been added to the SWBus providing functionality for logging and retrieving historic values of complex user-defined SWBus objects to disk. The module is available as a C++ library to be linked with the application program.

The values are named and time tagged. Variables may be added to the log at any time, and values or value attributes may be added, modified and retrieved. Data handled by the historic log module may be arbitrarily complex. However, care should be taken if the data involves pointers as pointers in general are not persistent.

Two different log types are supported, fixed size and fixed time span. The former type has a fixed number of entries and when filled, the oldest entry is overwritten, i.e. a ring buffer. The latter type has a fixed time span, thus the number of entries may vary. The buffer will automatically adapt to the necessary size. Entries falling outside the time span will be discarded.

In order to achieve an efficient implementation, all variables in a log must have the same data type. To log data of different data types, multiple logs must be created. The module implements a caching mechanism for adding values to the log in order to further enhance performance. The cache is emptied and its data written to disk whenever the cache is full and/or at periodic intervals.

This version of the historic log modules is restricted to single clients only. However, the application programmer's interface is prepared for multi-client access.

The various operational features include

- Open a new or delete an existing log.
- Add new variables to the log. Delete a single or all variables from the log.
- Add a variable entry. Clear all entries for a single or all variables in the log.
- Get the number of entries for a variable.
- Retrieve data for variable entries or entry attributes.
- Modify entries or entry attributes.
- Retrieve the time tags for a variable, all or within a given time interval.
- Get the number of variables in the log and retrieve the variable names.
- Make a copy or backup of the log.
- Get the last logged time, the log type, the time span or max entries of a variable.
- Check whether a variable exists in the log or not.
- Set the time span for fixed time span logs.
- Discard variable entries newer than a specified time.
- Suspend, restart or set another interval for the cache buffer flushing.

# Global Data Types

Description of global data types used in *SHL*.

---

Data Types

**TIMETAG** Struct data type used in *SHL*.

---

## Data Types

### TIMETAG

**Description**

**TIMETAG** is the time tag data type used in *SHL*. It is equal to the C struct **timeval**.

```
struct timeval
{
  time_t   tv_sec;
  long     tv_usec;
};
```

# HistoricLog Initialisation

The HistoricLog Initialisation function creates and initialises the internal data needed by the historic log module. It must be called after **SbInit** and prior to any of the methods of the class **HistoricLogClient**.

## Init Function

### initHistoricLog

```
int32 initHistoricLog()
```

**Description**

This function creates the **SWBus** classes that are used as interface between the **HistoricLogClient** class and the **HistoricLogServer** class.

**Access**

This function is public.

**Return Value**

When an error is detected it returns **SbCError**, else **SbCOK**.

**Example**

This example demonstrates how to initialise the log and start a logging session.

```
// The file myStruct.h contains the definition of
// the data type to be used in the log as shown below
struct TestData
{
  char   varString[20];
  int32  varInt;
  float  varFloat;
};


// The program code starts here
#include "HistoricLogClient.h"
#include "myStruct.h"

HistoricLogClient* myHL;

int32 main(int argc, char* argv[])
{
  char*  clientName     = "myClient";
  char*  fileName       = "myStruct.h";
  SbTSTI logProcess     = SbCPLocal;
  char*  clientName     = "testClient";
  char*  serverName     = "testServer";
  char*  logName        = "testLog";
  char*  logDirectory   = ".";
  char*  dataTypeName   = "TestData";
  int32  basicBlockSize = 100;
  int32  cacheSize      = 40;

  if (SbInit(clientName, NULL, NULL) != SbCOK)
  {
      printf("SbInit failed for process %s\n", clientName);
      exit(1);
  }
  if (initHistoricLog() != SbCOK)
  {
      printf("initHistoricLog failed for process %s\n", clientName);
      exit(1);
  }

  // Create SWBus class for the datatype to be used in the log
  if (SbReadScript(fileName) > 0)
  {
      printf("SbReadScript failed for file %s\n", fileName);
      exit(1);
  }

  myHL = new HistoricLogClient(logProcess, logName, logDirectory,
                               dataTypeName, basicBlockSize,
                               cacheSize, SHL FILE);

  if ( myHL->openLog( SHL READ WRITE ) == SbCError )
  {
      printf("Failed to open log\n");
      exit(1);
  }
  ...
}
```

**See Also:** **SbInit, HistoricLogClient::HistoricLogClient, openLog**

# HistoricLogClient

The C++ class **HistoricLogClient** is the interface class for the user of the historic log. It maintains the data and contains all necessary methods to utilise the log.

## Class Members

| **Attributes** | |
|---|---|
| **m_procId** | Process ident. |
| **m_objectId** | Object ident. |
| **m_dataSize** | Size of the data class in number of bytes. |
| **m_basicBlockSize** | Expansion size for dynamic-size logging. Given in number of entries. |
| **m_cacheSize** | Size of cache in percent of data block size. |
| **m_logMedium** | Log medium(file, etc.). |
| **m_logName** | The name of the log. |
| **m_logDirectory** | Path to the log directory. |
| **m_dataTypeName** | The name of the data class. |
| **m_periodicHandler** | Proxy for the periodic handler. |
| **m_currentInterval** | The current activation interval for the periodic handler. |
| **m_suspendedPeriodic** | Suspend flag for the periodic handler. |

| **Construction/Destruction** | |
|---|---|
| **HistoricLogClient** | Constructs **HistoricLogClient** object. |
| **~ HistoricLogClient** | Destroys **HistoricLogClient** object. |

| **Public Methods** | |
|---|---|
| **addEntry** | This method adds an entry to the log. |
| **addVariables** | This method adds a variable to the log. |
| **backupLog** | This method makes a backup copy of the log file. |
| **clearAll** | This method clears the logged for all variables. |
| **clearSingle** | This method clears the logged data for a named variable. |
| **copyLog** | This method copies the log to a named directory. |
| **closeLog** | This method closes the log. |
| **deleteAll** | This method deletes all variables in the log. |
| **deleteLog** | This method deletes the log. |
| **deleteSingle** | This method deletes a named variable in the log. |
| **deleteText** | This method deletes the number of entries in the cache. |

| | |
|---|---|
| **getEntries** | This method gets data from a named variable within a given time interval. |
| **getEntriesAttribute** | This method gets a data attribute from a named variable within a given time interval. |
| **getExistence** | This method checks the existence of a variable in the log. |
| **getLastLoggedTime** | This method gets the last logged time for a variable. |
| **getLogType** | This method gets the log type for a variable. |
| **getMaxEntries** | This method gets max entries for a variable. |
| **getNumberOfVariables** | This method gets the number of variables in the log. |
| **getNumEntries** | This method gets the number of entries for a variable. |
| **getTimeSpan** | This method gets the time span for a variable. |
| **getTimeTagEntries** | This method gets the time tags for a named variable within a given time interval. |
| **getTimeTags** | This method gets the time tags for a variable. |
| **getVariableNames** | This method gets the variable names in the log. |
| **modifyEntry** | This method modifies the data for a variable. |
| **modifyEntryAttribute** | This method modifies the data attribute for a variable. |
| **openLog** | This method makes an instance of the **SbCCHistoricLog** class and opens the log. |
| **setFlushInterval** | This method sets the cache flush interval. |
| **setTimeSpan** | This method sets the logging time span for a variable. |
| **timeAdjustAll** | This method sets the last time logged for all variables. |
| **timeAdjustSingle** | This method sets the last time logged for a variable. |

**Private Methods**

| | |
|---|---|
| **flushHandler** | This method is activated periodically to call the **flushData** method in the **HistoricLogSWBus** class. |

# Data Members

## HistoricLogClient::m_procId

**Description**

The attribute **m_procId** is the **SWBus** process ident for the server process.

**Access**

This attribute is protected.

# HistoricLogClient::m_objectId

**Description**

The attribute **m_objectId** is the object ident for the **SbCCHistoricLog** object.

**Access**

This attribute is protected.

# HistoricLogClient::m_dataSize

**Description**

The attribute **m_dataSize** is the size of the data class in number of bytes.

**Access**

This attribute is protected.

# HistoricLogClient::m_basicBlockSize

**Description**

The attribute **m_basicBlockSize** is the expansion size for dynamic-size logging(fixed time span). Given in number of entries.

**Access**

This attribute is protected.

# HistoricLogClient::m_cacheSize

**Description**

The attribute **m_cacheSize** is the size of the cache in percent of data block size.

**Access**

This attribute is protected.

# HistoricLogClient::m_logMedium

**Description**

The attribute **m_logMedium** is the log medium(file, etc.).

**Access**

This attribute is protected.

# HistoricLogClient::m_logName

**Description**

The attribute **m_logName** is the name of the log.

**Access**

This attribute is protected.

## HistoricLogClient::m_logDirectory

**Description**

The attribute **m_logDirectory** is the path to the log directory.

**Access**

This attribute is protected.

## HistoricLogClient::m_dataTypeName

**Description**

The attribute **m_dataTypeName** is the name of the data class.

**Access**

This attribute is protected.

## HistoricLogClient::m_periodicHandler

**Description**

The attribute **m_periodicHandler** is the proxy for the periodic handler.

**Access**

This attribute is protected.

## HistoricLogClient::m_currentInterval

**Description**

The attribute **m_currentInterval** is the current activation interval for the periodic handler.

**Access**

This attribute is protected.

## HistoricLogClient::m_suspendedPeriodic

**Description**

The attribute **m_suspendedPeriodic** is suspend flag for the periodic handler.

**Access**

This attribute is protected.

# Member Functions

## HistoricLogClient::HistoricLogClient

```
HistoricLogClient(       SbTSTI     processId,
                  const SbTString   logName,
                  const SbTString   logDirectory,
                  const SbTString   dataTypeName,
```

```
                        const int32      basicBlockSize,
                        const int32      cacheSize,
                        const int32      logMedium )
HistoricLogClient(            SbTSTI     processId,
                        const SbTString  logName,
                        const SbTString  logDirectory,
                              SbTSTI     dataTypeId,
                        const int32      basicBlockSize,
                        const int32      cacheSize,
                        const int32      logMedium )
```

This is the constructor for the **HistoricLogClient** class. It initialises the historic log client object.

**Access**

This constructor is public.

**Parameters**

*processId* Input parameter. The SWBus id of the process in which the log object shall be created.

*logName* Input parameter. The name of the log.

*logDirectory* Input parameter. The path of the log directory. The directory might be given absolute or relative. When the log server is running locally, the directory is relative to the directory of the local SWBus process. When the log server is running remotely, the directory is relative to the directory of the remote SWBus process.

*dataTypeName* Input parameter. The name of the SWBus class used for the variables in the log.

*dataTypeId* Input parameter. The SWBus id of the class used for the variables in he log.

*basicBlockSize* Input parameter. This argument is only applicable for fixed time span logs. In such logs the total number of entries may vary, and the historic log module allocates blocks for new entries when needed. The argument specifies the number of entries that will fit into each block.

*cacheSize* Input parameter. For fixed size logs: The size of the cache in percent of the maximum number of entries (specified as argument to the **addVariables** method). For fixed time span logs: The size of the cache in percent of the *basicBlockSize*. To ensure that data is written to file immediately the value 0 should be provided. In such cases, periodic flushing of the cache can be suspended, see **setFlushInterval**.

*logMedium* Input parameter. The log medium(this version supports file only). Legal value in this release is: **SHL_FILE**.

**Example**

See an example at **initHistoricLog**.

**See Also: inithistoricLog, openLog, addVariables, setFlushInterval**

# HistoricLogClient::~HistoricLogClient

```
~HistoricLogClient()
```

**Description**

This method is the destructor for the **HistoricLogClient** class. It destroys the **HistoricLogClient** object and frees the name buffers. It also removes the SWBus log object.

**Access**

This destructor is public.

# HistoricLogClient::addEntry

```
SbTSTI addEntry(const SbTString     variableName,
                      TIMETAG       timeTag,
                const SbTAddr       value ) const

SbTSTI addEntry(const SbTSTI        variableId,
                      TIMETAG       timeTag ) const
```

**Description**

Add a new entry for a specified variable.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The name of a variable in the log. |
| *variableId* | Input parameter. The id of a SWBus object. The object must have a name matching a variable name in the log and a class matching the class provided as input to the constructor of the **HistoricLogClient**. The value stored for the entry will be the current value of the SWBus object. |
| *timeTag* | Input parameter. Time tag for the entry. |
| *value* | Input parameter. Pointer to the value to be stored for the entry. The size of the data is determined by the data type provided as input to the constructor of the **HistoricLogClient** class. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

### Example

This example demonstrates how to add an entry to the log.

```
HistoricLogClient* myHL;

void myAddEntry()
{
  SbTString varName = ”name1”;
  TIMETAG   timeTag = {0, 0};
  TestData  value;

  timeTag.tv sec = time(0);
  strcpy(value.varString, "tore1");
  value.varInt = 20;
  value.varFloat = (float)3.14;
  if (myHL->addEntry(varName, timeTag, (SbTAddr)&value) == SbCError)
    printf("addEntry: Error return\n");

  return;
}
```

# HistoricLogClient::addVariables

```
SbTSTI addVariables(       int32    numberOfVariables,
                    const  SbTText  variableNames,
                           int32    logType,
                           int32    bufferSize ) const

SbTSTI addVariables(       int32    numberOfIds,
                    const  SbTSTI   variableIds,
                           int32    logType,
                           int32    bufferSize ) const
```

### Description

Add one or more variables to the log.

### Access

This function is public.

### Parameters

| | |
|---|---|
| *numberOfVariables* | Input parameter. The number of variables to add. |
| *numberOfIds* | Input parameter. The number of ids to add. |
| *variableNames* | Input parameter. The names of the variables to add. |
| *variable*Ids | Input parameter. The ids of the variables to add. |
| *logType* | Input parameter. Log type (fixed size or fixed time span). Legal values are: **SHL_BLOCK** for fixed size and **SHL_EXPAND** for fixed time span. Note that this argument implies that different variables in the log may have different log types provided that they are added by separate calls to **addVariables**. |
| *bufferSize* | Input parameter. If **logType** is fixed-size: The number of entries for each variable. If **logType** is fixed time span: The time span in seconds. |

### Return Value

When a local error is detected it returns SbCError, else the status from the log server process is returned. An error detected when processing one of the variables will result in an error return, even if the others were ok. This is because the error must be a system error and something is seriously bad.

**Example**

This example demonstrates how to add variables to the log.

```
HistoricLogClient* myHL;

void myAddVariables()
{
  int32     nVar = 3;
  char*     varNamePtr [] = {”name1”, ”name2”, ”name3”, NULL};
  SbTText   varNames = varNamePtr;
  int32     logType = 0;
  int32     blkSize = 100;

  if (myHL->addVariables(nVar, varNames, logType, blkSize) ==
SbCError)
      printf("addVariables: Error return\n");

return;
}
```

# HistoricLogClient::backupLog

**`SbTSTI backupLog() const`**

**Description**

Creates a backup file of the log by adding the extension “.back”.

**Access**

This function is public.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::clearAll

**`SbTSTI clearAll() const`**

**Description**

Clears the logged data for all variables.

**Access**

This function is public.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::clearSingle

**`SbTSTI clearSingle( const SbTString`** *`variableName`* **`) const`**

**Description**

Clears the logged data for a named variable.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The names of the variable to clear. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::copyLog

```
SbTSTI copyLog( const SbTString directoryName ) const
```

**Description**

Copies the log to a named directory.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *directoryName* | Input parameter. The directory to store the copy. The directory might be given absolute or relative. When the log server is running locally, the directory is relative to the directory of the local SWBus process. When the log server is running remotely, the directory is relative to the directory of the remote SWBus process. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::closeLog

```
SbTSTI closeLog( ) const
```

**Description**

Closes the log

**Access**

This function is public.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::deleteAll

```
SbTSTI deleteAll() const
```

**Description**

Deletes all variables from the log.

**Access**

This function is public.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::deleteLog

```
SbTSTI deleteLog() const
```

**Description**

Deletes the current log.

**Access**

This function is public.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::deleteSingle

```
SbTSTI deleteSingle( const SbTString variableName ) const
```

**Description**

Deletes a named variable from the log.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to delete from the log. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::deleteText

```
SbTSTI deleteText( int32      numberOfStrings,
                   SbTText&   text ) const
```

**Description**

Utility method used to delete data of type **SbTText** which is returned from the historic log.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *numberOfStrings* | Input parameter. The number of string pointers in the data type **SbTText**. |
| *text* | Input parameter. The data of type **SbTText**. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::getEntries

```
SbTSTI getEntries( const SbTString  variableName,
                   TIMETAG    timeTag,
                   int32&     numberOfEntries,
                   SbTAddr&   times,
                   SbTAddr&   values ) const

SbTSTI getEntries( const SbTSTI     variableId,
                   TIMETAG    timeTag,
                   int32&     numberOfEntries,
                   SbTAddr&   times,
                   SbTAddr&   values ) const

SbTSTI getEntries( const SbTString  variableName,
                   TIMETAG    timeTagStart,
                   TIMETAG    timeTagEnd,
                   int32&     numberOfEntries,
                   SbTAddr&   times,
                   SbTAddr&   values ) const

SbTSTI getEntries( const SbTSTI     variableId,
                   TIMETAG    timeTagStart,
                   TIMETAG    timeTagEnd,
                   int32&     numberOfEntries,
                   SbTAddr&   times,
                   SbTAddr&   values ) const
```

**Description**

Searches for data in the log with the given variable name and time tag or interval of time tags. The data and associated time tags are returned in allocated data areas. It is the responsibility of the user to delete the allocated areas when not needed.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to get entries from. |
| *variableId* | Input parameter. The variable id to get entries from. |
| *timeTag* | Input parameter. The time tag of the wanted data. |
| *timeTagStart* | Input parameter. The oldest time tag of the wanted data. |
| *timeTagEnd* | Input parameter. The newest time tag of the wanted data. |
| *numberOfEntries* | Output parameter. The number of entries found. |
| *times* | Output parameter. A pointer to the time tags associated with the found data. |
| *values* | Output parameter. A pointer to the found data. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

This example demonstrates how to get entries from the log.

```
HistoricLogClient* myHL;

void myGetEntries()
{
  int       ii;
  SbTString varName = ”name1”;
  TIMETAG   time1   = {0, 0};
  TIMETAG   time2   = {0, 0};
  int32     nEntries;
  SbTAddr   timeBuf;
  TIMETAG*  timeBufPtr;
  SbTAddr   valueBuf;
  testData* valueBufPtr;

  time1.tv sec = <time from>;
  time2.tv sec = <time to>;
  if (myHL->getEntries(varName, time1, time2,
                       nEntries, timeBuf, valueBuf) != SbCError)
  {
    valueBufPtr = (testData *)valueBuf;
    timeBufPtr  = (TIMETAG *)timeBuf;
    for (ii = 0; ii < nEntries; ii++)
    {
      printf("time: %d, string = %s, int = %d, float = %f\n",
             timeBufPtr[ii].tv sec, valueBufPtr[ii].varString,
             valueBufPtr[ii].varInt, valueBufPtr[ii].varFloat);
    }
    delete [] timeBuf;
    delete [] valueBuf;
  }
  else
    printf("getEntries: Error return\n");

  return;
}
```

# HistoricLogClient::getEntriesAttribute

```
SbTSTI getEntriesAttribute( const   SbTString   variableName,
                                    TIMETAG     timeTag,
                            const   SbTString   attributeName,
                                    int32&      numberOfEntries,
                                    SbTAddr&    times,
                                    SbTAddr&    values ) const

SbTSTI getEntriesAttribute( const   SbTSTI      variableId,
                                    TIMETAG     timeTag,
                            const   SbTString   attributeName,
                                    int32&      numberOfEntries,
                                    SbTAddr&    times,
                                    SbTAddr&    values ) const

SbTSTI getEntriesAttribute( const   SbTString   variableName,
                                    TIMETAG     timeTagStart,
                                    TIMETAG     timeTagEnd,
                            const   SbTString   attributeName,
                                    int32&      numberOfEntries,
                                    SbTAddr&    times,
                                    SbTAddr&    values ) const

SbTSTI getEntriesAttribute( const   SbTSTI      variableId,
                                    TIMETAG     timeTagStart,
```

```
                TIMETAG     timeTagEnd,
const           SbTString   attributeName,
                int32&      numberOfEntries,
                SbTAddr&    times,
                SbTAddr&    values ) const
```

**Description**

Searches for data attributes in the log with the given variable name, time tag or interval of time tags and attribute name. The data and associated time tags are returned in allocated data areas. It is the responsibility of the user to delete the allocated areas when not needed.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to get entries from. |
| *variableId* | Input parameter. The variable id to get entries from. |
| *timeTag* | Input parameter. The time tag of the wanted data. |
| *timeTagStart* | Input parameter. The oldest time tag of the wanted data. |
| *timeTagEnd* | Input parameter. The newest time tag of the wanted data. |
| *attributeName* | Input parameter. The attribute name of the data. |
| *numberOfEntries* | Output parameter. The number of entries found. |
| *times* | Output parameter. A pointer to the time tags associated with the found data. |
| *values* | Output parameter. A pointer to the found data. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

This example demonstrates how to get entries attributes from the log.

```
HistoricLogClient* myHL;

void myGetEntriesAttribute()
{
  int       ii;
  SbTString varName = ”name1”;
  SbTString attrName = ”varInt”;
  TIMETAG   time1   = {0, 0};
  TIMETAG   time2   = {0, 0};
  int32     nEntries;
  SbTAddr   timeBuf;
  TIMETAG*  timeBufPtr;
  SbTAddr   valueBuf;
  int32*    intPtr;

  time1.tv sec = <time from>;
  time2.tv_sec = <time to>;
  if (myHL->getEntriesAttribute(varName, time1, time2, attrName,
                       nEntries, timeBuf, valueBuf) != SbCError)
  {
    intPtr = (int32 *)valueBuf;
    timeBufPtr  = (TIMETAG *)timeBuf;
    for (ii = 0; ii < nEntries; ii++)
    {
      printf("time: %d, int = %d\n",
             timeBufPtr[ii].tv sec, intPtr[ii]);
    }
    delete [] timeBuf;
    delete [] valueBuf;
  }
  else
    printf("getEntriesAttribute: Error return\n");

  return;
}
```

# HistoricLogClient::getExistence

**SbTSTI getExistence( const SbTString** *variableName,*
**bool&** *existence* **) const**

**Description**

Method used to search for a variable name in the historic log.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to search for. |
| *existence* | Output parameter. True if the variable name is found in the log else false. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::getLastLoggedTime

```
SbTSTI getLastLoggedTime( const SbTString variableName,
                                TIMETAG&  lastLoggedTime ) const
```

**Description**

Method used to get the last logged time tag for a given variable.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to search for. |
| *lastLoggedTime* | Output parameter. The last logged time for the given variable. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::getLogType

```
SbTSTI getLogType( const SbTString  variableName,
                          int32&    logType ) const
```

**Description**

Method used to get the log type for a given variable.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to search for. |
| *logType* | Output parameter. The log type for the given variable. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::getMaxEntries

```
SbTSTI getMaxEntries( const SbTString     variableName,
                           int32&         maxEntries ) const
```

**Description**

Method used to get the max entries for a given variable.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to search for. |
| *maxEntries* | Output parameter. Max entries for the given variable. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::getNumberOfVariables

```
SbTSTI getNumberOfVariables( int32& numberOfVariables ) const
```

**Description**

Method used to get the number of variables in the log.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *numberOfVariables* | Output parameter. Number of variables in the log. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::getNumEntries

```
SbTSTI getNumEntries( const SbTString variableName,
                      int32&         numberOfEntries ) const

SbTSTI getNumEntries( const SbTSTI   variableId,
                      int32&         numberOfEntries ) const
```

**Description**

Method used to get the number of entries for a given variable.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to search for. |
| *variableId* | Input parameter. The variable id to search for. |
| *numberOfEntries* | Output parameter. Number of entries for the given variable. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::getTimeSpan

```
SbTSTI getTimeSpan( const SbTString variableName,
                    TIMETAG&       timeSpan ) const
```

**Description**

Method used to get the time span for a given variable. This applies for expandable logs only. If the variable is of the fixed-size type, SbCError is returned.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to search for. |
| *timeSpan* | Output parameter. The time span of the variable. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

---

# HistoricLogClient::getTimeTagEntries

```
SbTSTI getTimeTagEntries( const SbTString variableName,
                                TIMETAG   timeTagStart,
                                TIMETAG   timeTagEnd,
                                int32&    numberOfEntries,
                                SbTAddr&  values ) const
SbTSTI getTimeTagEntries( const SbTSTI    variableId,
                                TIMETAG   timeTagStart,
                                TIMETAG   timeTagEnd,
                                int32&    numberOfEntries,
                                SbTAddr&  values ) const
```

**Description**

Searches for the time tags in the log within the given time tag interval. The time tags are returned in an allocated data area. It is the responsibility of the user to delete the allocated area when not needed.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to get time tags from. |
| *variableId* | Input parameter. The variable id to get time tags from. |
| *timeTagStart* | Input parameter. The oldest time tag of the interval. |
| *timeTagEnd* | Input parameter. The newest time tag of the interval. |
| *numberOfEntries* | Output parameter. The number of entries found. |
| *values* | Output parameter. A pointer to the time tags within the interval. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

This example demonstrates how to get time tag entries from the log.

```
HistoricLogClient* myHL;

void myGetTimeTagEntries()
{
  int       ii;
  SbTString varName = ”name1”;
  TIMETAG   time1   = {0, 0};
  TIMETAG   time2   = {0, 0};
  int32     nEntries;
  SbTAddr   timeBuf;
  TIMETAG*  timeBufPtr;
  char      calendar[40];
  char*     cFormat = "%H:%M:%S";

  time1.tv sec = <time from>;
  time2.tv sec = <time to>;
  if (myHL->getTimeTagEntries(varName, time1, time2,
                              nEntries, timeBuf) != SbCError)
  {
    timeBufPtr = (TIMETAG *)timeBuf;
    for (ii = 0; ii < nEntries; ii++)
    {
      strftime(calendar, 40, cFormat,
               localtime(&timeBufPtr [ii].tv sec));
      printf("timeTag = %d  %s\n", timeBufPtr[ii].tv sec, calendar);

    }
    delete [] timeBuf;
  }
  else
    printf("getTimeTagEntries: Error return\n");

  return;
}
```

# HistoricLogClient::getTimeTags

```
SbTSTI getTimeTags( const SbTString variableName,
                          int32&    numberOfEntries,
                          SbTAddr&  timeTags ) const

SbTSTI getTimeTags( const SbTSTI    variableId,
                          int32&    numberOfEntries,
                          SbTAddr&  timeTags ) const
```

**Description**

Searches for all time tags in the log for the given variable. The time tags are returned in an allocated data area. It is the responsibility of the user to delete the allocated area when not needed.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to get time tags from. |
| *variableId* | Input parameter. The variable id to get time tags from. |
| *numberOfEntries* | Output parameter. The number of entries found. |
| *timeTags* | Output parameter. A pointer to the time tags. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

This example demonstrates how to get time tag entries from the log.

```
HistoricLogClient* myHL;

void myGetTimeTags()
{
  int        ii;
  SbTString  varName = ”name1”;
  int32      nEntries;
  SbTAddr    timeBuf;
  TIMETAG*   timeBufPtr;
  char       calendar[40];
  char*      cFormat = "%H:%M:%S";

  if (myHL->getTimeTags(varName, nEntries, timeBuf) != SbCError)
  {
    timeBufPtr  = (TIMETAG *)timeBuf;
    for (ii = 0; ii < nEntries; ii++)
    {
      strftime(calendar, 40, cFormat,
               localtime(&timeBufPtr[ii].tv sec));
      printf("timeTag = %d  %s\n", timeBufPtr[ii].tv sec, calendar);

    }
    delete [] timeBuf;
  }
  else
    printf("getTimeTags: Error return\n");

  return;
}
```

# HistoricLogClient::getVariableNames

```
SbTSTI getVariableNames( int32&     numberOfVariables,
                         SbTText&   variableNames ) const

SbTSTI getVariableNames( int32&     numberOfVariables,
                         SbTSTI&    variableIds ) const
```

**Description**

Gets the variable names in the log. The variable names are returned in an allocated data area. It is the responsibility of the user to delete the allocated area when not needed.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *numberOfVariables* | Output parameter. The number of variables in the log. |
| *variableNames* | Output parameter. A pointer to the variable names. |
| *variableIds* | Output parameter. A pointer to the variable ids. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

This example demonstrates how to get variable names from the log.

```
HistoricLogClient* myHL;

void myGetVariableNames()
{
  int       ii;
  SbTText   varNames;
  int32     nVar;

  if (myHL->getVariableNames(nVar, varNames) != SbCError)
    for (ii = 0; ii < nVar && varNames[ii] != NULL; ii++)
      printf("%s\n", varNames[ii]);
  else
    printf("getVariableNames: Error return\n");

  if (varNames != NULL)
    myHL->deleteText(nVar, varNames); //Delete the returned text.

  return;
}
```

**See Also:** **deleteText**

---

# HistoricLogClient::modifyEntry

```
SbTSTI modifyEntry( const SbTString variableName,
                          TIMETAG   timeTag,
                    const SbTAddr   value ) const

SbTSTI modifyEntry( const SbTSTI    variableId,
                          TIMETAG   timeTag ) const
```

**Description**

Modifies the data in the log for the given variable and time tag.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The name of the variable to be modified. |
| *variableId* | Input parameter. The id of the variable to be modified. See more detailed description at the function addEntry. |
| *timeTag* | Input parameter. The time tag for the entry to be modified. |
| *value* | Input parameter. Pointer to the new data. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

This example demonstrates how to modify an entry in the log.

```
typedef struct
{
char  varString[20];
int32 varInt;
float varFloat;
}TestData;

HistoricLogClient* myHL;

void myModifyEntry()
{
  SbTString varName = ”name1”;
  TIMETAG   timeTag = {0, 0};
  TestData  value;

  timeTag.tv sec = <time to modify>;
  strcpy(value.varString, "fghij");
  value.varInt = 3;
  value.varFloat = (float)9.99;
  if (myHL->modifyEntry(varName, timeTag, (SbTAddr)&value)==
      SbCError)
    printf("modifyEntry: Error return\n");

  return;
}
```

**See Also:** **addEntry**

---

# HistoricLogClient::modifyEntryAttribute

```
SbTSTI modifyEntryAttribute(const SbTString variableName,
                                   TIMETAG   timeTag,
                            const SbTString attributeName,
                            const SbTAddr   value ) const

SbTSTI modifyEntryAttribute(const SbTSTI    attributeId,
                                   TIMETAG   timeTag ) const
```

**Description**

Modifies the attribute in the log for the given variable, time tag and attribute name.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The name of the variable to be modified. |
| *attributeId* | Input parameter. The id of the attribute to be modified. |
| *timeTag* | Input parameter. The time tag for the entry to be modified. |
| *attributeName* | Input parameter. The name of the attribute to be modified. |
| *value* | Input parameter. Pointer to the new attribute data. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

This example demonstrates how to modify an entry attribute in the log.

```
HistoricLogClient* myHL;

void myModifyEntryAttribute()
{
  SbTString varName  = ”name1”;
  SbTString attrName = ”varInt”;
  TIMETAG   timeTag  = {0, 0};
  int32     varInt;

  timeTag.tv_sec = <time to modify>;
  varInt = 88;
  if (myHL->modifyEntryAttribute(varName, timeTag, attrName,
                                 (SbTAddr)&varInt) == SbCError)
    printf("modifyEntryAttribute: Error return\n");

  return;
}
```

# HistoricLogClient::openLog

**SbTSTI openLog( int32** *accessMode* **)**

**Description**

Opens the log (create new or open existing) with the specified access mode. Creates a SWBus object named by the *logName* argument of the constructor of **HistoricLogClient**. The SWBus object is of class **SbCCHistoricLog** and is located in the process denoted by the *processId* argument of the constructor of **HistoricLogClient**.

This method must be called after the **initHistoricLog** function but prior to any other method of the **HistoricLogClient** class. Further, prior to calling this method the SWBus class for the data to be added to the log must exist. The name of this SWBus class must match the argument dataTypeName of the **HistoricLogClient** constructor.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *AccessMode* | Input parameter. The access mode of the log. Legal values are: **SHL_READ**, **SHL_WRITE** and **SHL_READ_WRITE**. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

**Example**

See an example at **initHistoricLog**.

**See Also: initiHistoricLog, HistoricLogClient::HistoricLogClient**

# HistoricLogClient::setFlushInterval

```
SbTSTI setFlushInterval( const TIMETAG& interval )
```

**Description**

Sets the interval to be used for flushing the cache contents to disk. If interval is 0, periodic flushing is suspended. If interval is bigger than 0, the cache is flushed periodically with the specified interval. In any circumstances the cache is flushed whenever it is full.

**Access**

This function is public.

**Parameters**

*interval* Input parameter. The flush interval.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::setTimeSpan

```
SbTSTI setTimeSpan( const SbTString variableName,
                          TIMETAG&  timeSpan ) const
```

**Description**

Set a new time span in the log for the given variable. This applies for expandable logs only. If the variable is of the fixed-size type, SbCError is returned.

**Access**

This function is public.

**Parameters**

*variableName* Input parameter. The variable name to search for.

*timeSpan* Input parameter. The new time span.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::timeAdjustAll

```
SbTSTI timeAdjustAll( const TIMETAG&  time ) const
```

**Description**

Discard all entries in the log newer than the specified time.

**Access**

This function is public.

**Parameters**

*time* Input parameter. The new log time.

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.

# HistoricLogClient::timeAdjustSingle

```
SbTSTI timeAdjustSingle( const SbTString variableName,
                                 TIMETAG&  time ) const
```

**Description**

For the specified variable: discard all entries newer than the specified time.

**Access**

This function is public.

**Parameters**

| | |
|---|---|
| *variableName* | Input parameter. The variable name to search for. |
| *time* | Input parameter. The new log time. |

**Return Value**

When a local error is detected it returns SbCError, else the status from the log server process is returned.