



Product:	CDP2SQL
Product version:	v1.0
Document ID:	UM-CDP2SQL
Doc revision:	A
Written/Appr.:	NPE / KD
Date:	12. Feb. 2009

## Industrial Control Design AS



# CDP2SQL v1.0

## User Manual

The content of this document is confidential information not to be published without the consent of Industrial Control Design AS.

Industrial Control Design AS, [www.icd.no](http://www.icd.no), [support@icd.no](mailto:support@icd.no), [forum.icd.no](http://forum.icd.no)

# Contents

<b>1. INTRODUCTION.....</b>	<b>4</b>	<b>3. APPLICATION CONFIGURATION.....</b>	<b>9</b>
1.1. About.....	4	3.1. About.....	9
1.2. Overview.....	4	3.2. How to add the SQL Logger component to a CDP Application.....	9
1.2.1. The CDP2SQL Application.....	4		
1.2.2. The SQL Logger Component.....	4		
1.2.3. The SQL Logger State Machine.....	5		
<b>2. SQL LOGGER FEATURES.....</b>	<b>6</b>	<b>4. CONFIGURATION.....</b>	<b>10</b>
2.1. About.....	6	4.1. About.....	10
2.2. Periodic logging.....	6	4.2. Activate.....	10
2.2.1. Description.....	6	4.2.1. Description.....	10
2.2.2. Table Example.....	6	4.2.2. Example.....	10
2.3. Event logging.....	6	4.3. fs.....	10
2.3.1. Description.....	6	4.3.1. Description.....	10
2.3.2. Table Example.....	7	4.3.2. Example.....	10
2.4. Command logging.....	7	4.4. Database.....	10
2.4.1. Description.....	7	4.4.1. Description.....	10
2.4.2. Table Example.....	7	4.4.2. Example.....	10
2.5. Delta logging.....	7	4.4.3. Attributes.....	11
2.5.1. Description.....	7	4.5. LogControl.....	11
2.5.2. Table Example.....	7	4.5.1. Description.....	11
2.6. Burst logging.....	8	4.5.2. Example.....	11
2.6.1. Description.....	8	4.5.3. Attributes.....	11
2.7. Query to file.....	8	4.6. SignalLogging.....	12
2.7.1. Description.....	8	4.6.1. Description.....	12
2.7.2. File Example.....	8	4.6.2. Example.....	12
2.8. Table Overflow.....	8	4.6.3. Attributes.....	12
2.8.1. Description.....	8	4.7. BufferOptions.....	12
2.8.2. Overflow Methods.....	8	4.7.1. Description.....	12
2.9. Temporary buffers.....	8	4.7.2. Example.....	12
2.9.1. Description.....	8	4.7.3. Attributes.....	12
		4.8. TableOptions.....	12
		4.8.1. Description.....	12
		4.8.2. Example.....	12
		4.8.3. Attributes.....	13
		4.9. BurstOptions.....	13
		4.9.1. Description.....	13
		4.9.2. Example.....	13
		4.9.3. Attributes.....	13
		4.10. EventLogging.....	13
		4.10.1. Description.....	13
		4.10.2. Example.....	13
		4.10.3. Attributes.....	13
		4.11. QueryRequest.....	14
		4.11.1. Description.....	14
		4.11.2. Example.....	14
		4.11.3. BufferOptions Attributes.....	14
		4.11.4. FileOptions Attributes.....	14

4.12. SQLCommand.....	14
4.12.1. Description.....	14
4.12.2. Example.....	14
4.12.3. Attributes.....	14
4.13. InfoTable.....	15
4.13.1. Description.....	15
4.13.2. Example.....	15
4.13.3. Attributes.....	15
4.13.4. Line Attributes.....	15
4.14. SignalGroup.....	15
4.14.1. Description.....	15
4.14.2. Example.....	15
4.15. SignalSource.....	15
4.15.1. Description.....	15
4.15.2. Example.....	15
4.15.3. Attributes.....	16
4.16. Signals.....	16
4.17. Alarms.....	16
4.18. Parameters.....	16

---

## 5. DEMO APPLICATIONS.....17

5.1. About.....	17
5.2. Overview.....	17
5.2.1. Application overview.....	17

5.2.2. The LogManager.....	17
5.2.3. The LoggerApp.....	17
5.2.4. Demo directory structure.....	17
5.3. Using the LogManager to control the LoggerApp.....	18
5.3.1. Getting started.....	18
5.3.2. Log control by signal.....	18
5.3.3. Log control by message.....	18
5.3.4. Sending of event messages.....	19
5.3.5. Sending of query messages.....	19
5.3.6. Sending of SQL Command messages.....	20
5.3.7. Controlling the log frequency.....	20

---

## 6. DATA EXTRACTION.....21

6.1. Command-line database clients.....	21
6.2. Database GUI browsers.....	21
6.3. Scripting language bindings.....	21
6.3.1. Python and SQLite3 example code.....	21
6.3.2. Ruby and SQLite3 example code.....	21

---

## 7. APPENDIX.....22

7.1. Example Component File.....	22
----------------------------------	----

# 1. Introduction

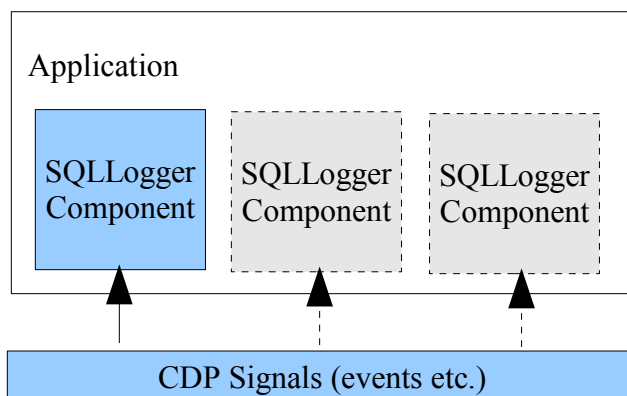
## 1.1. About

This document describes the SQL Logger component of the CDP2SQL library. Details will be provided to make the reader understand how the component works, and how to configure and use it with the CDP system. For information regarding the CDP2SQL API, see the programmers manual.

## 1.2. Overview

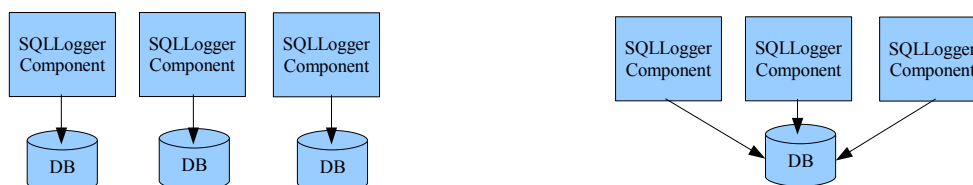
### 1.2.1. The CDP2SQL Application

The SQL Logger component logs a group of signals every LogFs (a configurable attribute in the component XML file). Every instance of the SQL Logger must have a separate XML file with configuration, like table name, database name, signals etc. The SQL Logger component uses the CDP2SQL API to access the database. When multiple database interfaces exist, the database type is selected in XML along with the other configuration.



### 1.2.2. The SQL Logger Component

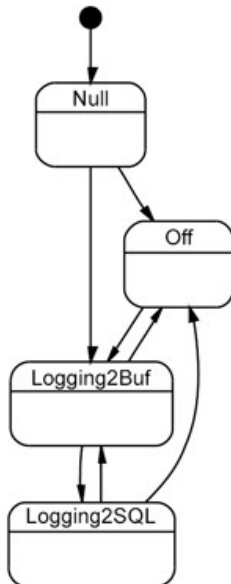
Every instance of the SQL Logger can share the same database, or they can have an unique database of their own. The latter is recommended as this will ensure that limitation related to concurrency won't reduce the determinism of logging. To enable an unique database for each instance, simply make the DBName tag in component XML unique for each logger instance. Note that each instance must have an unique component XML file.



### 1.2.3. The SQL Logger State Machine

The state machine of the SQL Logger component includes 4 states. The logger is started in 'Logging2Buf' state. In this state no data is added to the database, but to temporary buffers.

Then, when logging is enabled by either message ('Logging2SQL' text command) or the LogControl signal, the temporary buffers are added to the database and data is continuously added each LogFs. In the event of an error, the logger goes to the 'Off' state.



## 2. SQL Logger Features

### 2.1. About

This chapter describes the various SQL Logger features and logging types.

### 2.2. Periodic logging

#### 2.2.1. Description

Each logger component can be set up with a group of signals that are logged periodically. The resulting table rows will include an unique id and time stamp followed by the actual signal values. Column names are identical to that of the logged signal.

#### 2.2.2. Table Example

The following table was created by a logger component named LoggerOne, containing a signal group with two signals named FirstSig (double) and SecondSig (int). As the timestamps indicate, the logging frequency was set to 1Hz.

id	timestamp	FirstSig	SecondSig
1	2009-01-29 15:41:11.119	0.3	1
2	2009-01-29 15:41:12.119	0.5	2
3	2009-01-29 15:41:13.119	0.5	2
4	2009-01-29 15:41:14.119	0.7	2
5	2009-01-29 15:41:15.119	0.6	4

### 2.3. Event logging

#### 2.3.1. Description

The logger component can receive and store event data as described in the below example. Typical use would be to add an event subscriber in the same application as the logger, and make the subscriber forward event data. For information about sending event messages to the logger, see the programmers manual. Further details regarding the event subscriber can be found in the documentation for the CDPEventManagerLib.

### 2.3.2. Table Example

The following table has been filled with dummy values to show the various column names.

id	LogStamp	Event Id	Original EventId	Handle	Status	Event Code	Timestamp	EventName	EventDescription	ObjectName
1	2009-01-29 15:41:11.119	23	23	1	0	2	Mon Jan 19 12:07:03 2009	StateChanged	State transition	StateMaster
2	2009-01-29 15:41:11.119	34	34	2	8	1	Mon Jan 19 14:21:34 2009	StateChanged	Dangerous alarm	MotorApp
3	2009-01-29 15:41:11.119	35	35	3	1	8	Mon Jan 19 21:03:23 2009	LogMessage	Custom message	Switch

## 2.4. Command logging

### 2.4.1. Description

The command logging feature enables users to create tables and log data using standard SQL commands. This provides users with freedom to create indexes as needed and perform custom log styles like incrementing rows.

### 2.4.2. Table Example

id	dataOne	dataTwo
1	My own custom table!	I can create an index on this column if I want to..
2	I can put anything I want in here..	I can create as many column as I want to..
3	I can update rows when required..	I can set any column type as long as it is supported..

## 2.5. Delta logging

### 2.5.1. Description

With delta logging enabled, a signal needs to change more than a specified delta value to get logged. In the logger component however, this has been implemented in such way that all samples are logged if a single signal changes more than its delta value.

### 2.5.2. Table Example

The following table shows how the table in section 2.2.2 would have looked like with delta logging enabled. The delta value of SecondSig is set to 0.5 while FirstSig has a value of 0.1.

id	timestamp	FirstSig	SecondSig
1	2009-01-29 15:41:11.119	0.3	1
2	2009-01-29 15:41:12.119	0.5	2
3	2009-01-29 15:41:14.119	0.7	2
4	2009-01-29 15:41:15.119	0.6	4

## 2.6. Burst logging

### 2.6.1. Description

Log data at a high frequency for a short period of time. Both frequency and time period is configurable in XML as described in section 4.9. Burst logging is executed by sending the `BurstLogging` message.

## 2.7. Query to file

### 2.7.1. Description

The SQL Logger accepts query strings per message and outputs the results to a file as configured in XML. It is also possible to specify a different file name and delimiter by adding this information in the message. See the programmers manual for more information.

### 2.7.2. File Example

```
Timestamp.....: Thu Jan 29 15:26:22.299 2009
Query.....: select * from LoggerOne
Delimiter.....: ','
Description...: Column names followed by rows with column data.

id,timestamp,FirstSig,SecondSig
1,2009-01-29 15:24:32.091,0.1,2
2,2009-01-29 15:24:33.091,0.5,4
3,2009-01-29 15:24:34.091,0.5,5
4,2009-01-29 15:24:35.091,1.4,4
```

## 2.8. Table Overflow

### 2.8.1. Description

Table overflow can be handled by one of three configurable methods. See the following section for details.

### 2.8.2. Overflow Methods

#	Method name	Description
0	Continue	Raise an alarm, but continue logging.
1	Stop	Raise an alarm and stop logging.
2	Circular Buffer	Raise an alarm and continue logging in round robin style, overwriting old samples.

## 2.9. Temporary buffers

### 2.9.1. Description

The SQL Logger uses configurable circular buffers in all logger operations. This prevents concurrency problems and lost samples while the database is busy or blocked. In addition it makes it possible to perform history logging as the logger adds samples to the buffer before entering the logging state. Thus, if the logger only enters logging state triggered by an external event, it is possible to get samples from before the external event occurred. See section 4.7.3 for more details.

# 3. Application Configuration

## 3.1. About

This chapter describes how to instantiate the SQL Logger component within a CDP application.

## 3.2. How to add the SQL Logger component to a CDP Application

Add the following to your project's Application.xml:

Inside the <Components> element, add an instance of a SQL Logger component, for instance:

```
<Component Name="SQLLogger_1" Type="SQLLogger" src="Components/SQLLogger_1.xml"></Component>
```

Or, inside the <Subcomponents> element, add:

```
<Subcomponent Name="SQLLogger_1" Type="SQLLogger" src="Components/SQLLogger_1.xml"> </Subcomponent>
```

This will tell CDP to initialize a component named "SQLLogger\_1" from a component file located at "Components/SQLLogger\_1.xml". Make sure that your Models folder contains a SQLLogger.xml model file, or the component will not be initialized correctly.

Configuration is done by modifying the component xml file. It should not be necessary to modify the model XML file. An example of SQLLogger.xml file (component XML) is found in 7.1.

# 4. Configuration

## 4.1. About

This chapter describes the various XML configuration parameters of the SQL Logger component.

## 4.2. Activate

### 4.2.1. Description

Specifies the time to delay startup of the component, after configure has been run.

### 4.2.2. Example

```
<Activate>1</Activate>
```

## 4.3. fs

### 4.3.1. Description

This tag sets the rate in Hertz at which the process function of the component is executed. Note that the component signals are requested at the same rate as 'fs'. Thus, strive to keep 'fs' as low as possible to avoid unnecessary network traffic.

The SQL Logger automatically increases 'fs' if it is lower than the signal log frequency. Be aware that 'fs' is never decreased while the application is running. This is due to the fact that the signal request rate, controlled by the MessengerIOserver, can only be increased after startup. Thus, to decrease network traffic the application has to be restarted after adjusting 'fs' to a lower value.

### 4.3.2. Example

```
<fs>1</fs>
```

## 4.4. Database

### 4.4.1. Description

This element contains attributes related to database access.

### 4.4.2. Example

```
<Database DBName="cdp2sql.db" Type="sqlite3"></Database>
```

### 4.4.3. Attributes

Attribute Name	Description
DBName	Insert the name of the database here. Note that when running multiple instances of the SQLLogger, it might be wise to have an unique database for each instance. This ensures that the database is not locked by other threads when you need access. Note that the database is created if the specified DB does not exist.
Type	This attribute specifies the database type you wish to execute. Currently, the only supported type is 'sqlite3'.

## 4.5. LogControl

### 4.5.1. Description

Element with attributes related to data logging.

### 4.5.2. Example

```
<LogControl TimeFormat="0" TransactionRetry="5">
  <SignalLogging LogFs="0.2" LogControlFs="1">
    <BufferOptions BufferSize="1000" BufferKeep="100" BufferFlush="40"></BufferOptions>
    <TableOptions MaxRows="999" OverflowMethod="2" OverwriteTable="1"></TableOptions>
    <BurstOptions BurstFs="10" BurstPeriodMs="2000"></BurstOptions>
  </SignalLogging>
  <EventLogging>
    <BufferOptions BufferSize="100" BufferKeep="100" BufferFlush="40"></BufferOptions>
    <TableOptions MaxRows="999" OverflowMethod="2" OverwriteTable="1"></TableOptions>
  </EventLogging>
  <QueryRequest>
    <BufferOptions BufferSize="3"></BufferOptions>
    <FileOptions OutputFile="query.txt" Delimiter=","></FileOptions>
  </QueryRequest>
  <SQLCommand>
    <BufferOptions BufferSize="100" BufferKeep="100" BufferFlush="3"></BufferOptions>
  </SQLCommand>
</LogControl>
```

### 4.5.3. Attributes

Attribute Name	Description
TimeFormat	<p>The format of the timestamp that will be added along with the logged data. To achieve the fastest possible log times, set the attribute to '1'. This enables timestamps of type double, with values representing the time since epoch in seconds. The default and more human readable format is ISO 8601 strings of type yyyy-mm-dd hh:mm:ss.ms (e.g. 2008-12-24 16:14:13.013). This format is enabled when TimeFormat is set to '0'.</p> <p>Note that when the 'TimeFormat' is changed, new tables has to be created for the timestamp to match the table type. To erase the old tables, set 'OverwriteTable' to '1' (for both SignalLogging and EventLogging) and restart the application. Remember to backup any old log data before doing this, and be careful to set 'OverwriteTable' back to '0' after starting the application. Else, any logged data will get overwritten every time the app is restarted.</p>
TransactionRetry	<p>All database access that changes the database contents are performed in transactions to improve performance and protect data. When a transaction fails and is rolled back, the TransactionRetry attribute specifies how many times to retry executing the transaction. Note that at this point, data samples have already been popped from the buffer. Thus, a rolled back transaction without retry will lead to lost samples. Also note that a high TransactionRetry value requires a large buffers to ensure that the buffers doesn't run full while the database is busy with the transaction retries (e.g. retries due to blocked database might keep a transaction waiting for more than a minute before failing).</p>

## 4.6. SignalLogging

### 4.6.1. Description

Element with attributes related to signal logging.

### 4.6.2. Example

```
<SignalLogging LogFs="0.2" LogControlFs="1">
  <BufferOptions BufferSize="1000" BufferKeep="100" BufferFlush="40"></BufferOptions>
  <TableOptions MaxRows="999" OverflowMethod="2" OverwriteTable="1"></TableOptions>
  <BurstOptions BurstFs="10" BurstPeriodMs="2000"></BurstOptions>
</SignalLogging>
```

### 4.6.3. Attributes

Attribute Name	Description
LogFs	The signal logging frequency. Note that a high frequency will make the logger increase its own 'fs' to obtain updated signal samples. This will lead to increased network traffic.
LogControlFs	When this attribute is set to '1', the LogControl signal is used both to enable logging and to control the frequency of the periodic logging process.

## 4.7. BufferOptions

### 4.7.1. Description

Element with attributes related to a circular buffer.

### 4.7.2. Example

```
<BufferOptions BufferSize="1000" BufferKeep="100" BufferFlush="40"></BufferOptions>
```

### 4.7.3. Attributes

Attribute Name	Description
BufferSize	The size of the circular buffer that is used to store samples while the database is busy, or while running in the 'Logging2Buf' state. Setting 'BufferSize' to '0' disables logging. Note that all samples in the buffer are written to the database when entering the 'Logging2SQL' state. To prevent this from happening, use the 'BufferKeep' attribute to specify how many of the samples to keep.
BufferKeep	In the transition from 'Logging2Buf' to 'Logging2SQL', the default logger behaviour is to write all buffered samples to the database. For a shorter buffer history, set attribute 'BufferKeep'. Be aware that setting 'BufferKeep' to '0' will make the logger drop all samples, while a number identical to 'BufferSize' will make the logger keep all samples.
BufferFlush	How many samples from the buffer should be added to the database in a single transaction (too many samples might decrease performance due to memory constraints).

## 4.8. TableOptions

### 4.8.1. Description

Element with attributes related to a database table.

### 4.8.2. Example

```
<TableOptions MaxRows="999" OverflowMethod="2" OverwriteTable="1"></TableOptions>
```

### 4.8.3. Attributes

Attribute Name	Description
MaxRows	The maximum number of rows in the table. Make a rough size estimate based on number of components, signals, events and disk space.
OverflowMethod	Set this attribute to '1' to stop logging when the maximum number of table rows is reached. To continue logging in round robin style, set the attribute to '2'. It is also possible to continue logging as usual by setting the attribute to '0'. Note that this might be dangerous as it is impossible to tell when the application will run out of disk space. In any case, the component will raise an alarm when 'MaxRows' is reached.
OverwriteTable	Set this attribute to '1' only when you want to overwrite an existing table. <b>Be careful to set it back to '0' if you don't want to overwrite logged data every time the Application is restarted!</b>

## 4.9. BurstOptions

### 4.9.1. Description

Element with attributes related to burst logging.

### 4.9.2. Example

```
<BurstOptions BurstFs="10" BurstPeriodMs="2000"></BurstOptions>
```

### 4.9.3. Attributes

Attribute Name	Description
BurstFs	<p>The signal logging frequency during burst logging. Note that component 'fs' will get increased if burst logging is executed with 'BurstFs' &gt; 'fs'. This will make the MessengerIOserver request packets at a higher rate which in turn results in higher network load. The application has to be restarted to reset packet requests to the initial rate.</p> <p>In addition, it is important to be aware that 'BurstFs' &gt; 'fs' might lead to faulty signal samples the first time burst logging is executed. This is due to the fact that there will be a delay from detecting the change in 'fs' and up to the point where signals are actually received at the new 'fs'. Thus, strive to make 'BurstFs' identical to the component 'fs'.</p>
BurstPeriodMs	The period of time that the component will log samples at the rate specified by 'BurstFs'.

## 4.10. EventLogging

### 4.10.1. Description

Element with attributes related to event logging.

### 4.10.2. Example

```
<EventLogging>
  <BufferOptions BufferSize="100" BufferKeep="100" BufferFlush="40"></BufferOptions>
  <TableOptions MaxRows="999" OverflowMethod="2" OverwriteTable="1"></TableOptions>
</EventLogging>
```

### 4.10.3. Attributes

See section 4.7 and 4.8 for information regarding BufferOptions and TableOptions.

## 4.11. QueryRequest

### 4.11.1. Description

Element with attributes related to query requests.

### 4.11.2. Example

```
<QueryRequest>
  <BufferOptions BufferSize="3"></BufferOptions>
  <FileOptions OutputFile="query.txt" Delimiter=","></FileOptions>
</QueryRequest>
```

### 4.11.3. BufferOptions Attributes

Attribute Name	Description
BufferSize	<p>The one and only attribute for the query buffer. Since queries does not change the database contents, it is possible to execute incoming queries even when in the 'Logging2Buf' state. Other than that, the attribute is identical to that described in section 4.7.3.</p> <p>Note that 'BufferFlush' attribute is also missing from the BufferOptions. This is due to the fact that queries are only executed one at the time, and does thus not require a transaction.</p>

### 4.11.4. FileOptions Attributes

Attribute Name	Description
OutputFile	The default name of the output file containing the query result. Note that this attribute is only used when the incoming query request does not specify a valid output file name.
Delimiter	The default delimiter used to separate values in the query result. Note that this attribute is only used when the incoming query request does not specify a valid delimiter.

## 4.12. SQLCommand

### 4.12.1. Description

Element with attributes related to SQL command logging.

### 4.12.2. Example

```
<SQLCommand>
  <BufferOptions BufferSize="100" BufferKeep="100" BufferFlush="3"></BufferOptions>
</SQLCommand>
```

### 4.12.3. Attributes

See section 4.7 for information regarding BufferOptions.

## 4.13. InfoTable

### 4.13.1. Description

The InfoTable element can contain additional information about the signal group that is to be logged. The data will be added in a separate table alongside the actual logging table. While the default name of the logging table is identical to the component name, the default name of the information table is the component name followed by '\_Info'. Note that any information can be inserted here as long as it is added in the format described in the following example.

### 4.13.2. Example

```
<InfoTable Overwrite="0" ForceName="">
  <Line Name="Date" Value=""></Line>
  <Line Name="Time" Value=""></Line>
  <Line Name="Logging" Value="MANUAL"></Line>
  <Line Name="IMO Number" Value="IMO9181065"></Line>
  <Line Name="Unit Number" Value="xxxxxxxxx"></Line>
  <Line Name="Signal Type" Value="VIBRATION"></Line>
  <Line Name="Pitch" Value="80"></Line>
</InfoTable>
```

### 4.13.3. Attributes

Attribute Name	Description
Overwrite	Set this attribute to '1' to overwrite existing table at startup.
ForceName	Insert table name here if wanting to use something other than the default 'ComponentName_Info'.

### 4.13.4. Line Attributes

Attribute Name	Description
Name	A name or title that describes what is to come in the value field.
Value	The value or text that was described by the name field.

## 4.14. SignalGroup

### 4.14.1. Description

Add all signals that are to be logged as illustrated in the following example.

### 4.14.2. Example

```
<SignalGroup>
  <SignalSource Name="FirstSig" Type="double" DeltaValue="1.0" Routing="Thruster1.Temp"
    Description="First signal."></SignalSource>
  <SignalSource Name="SecondSig" Type="int" DeltaValue="1" Routing="Thruster2.Temp"
    Description="Second signal."></SignalSource>
</SignalGroup>
```

## 4.15. SignalSource

### 4.15.1. Description

A signal to be logged as part of a SignalGroup.

### 4.15.2. Example

```
<SignalSource Name="FirstSig" Type="int" DeltaValue="1" Routing="Thruster1" Description="A signal.">
</SignalSource>
```

### 4.15.3. Attributes

Attribute Name	Description
Name	The signal name, which will also be used as column name in the signal logging table.
Type	Signal data type that will get converted to valid SQL before creating the table. Valid options include 'double', 'int', and 'bool'.
DeltaValue	When the 'EnableDeltaLogging' parameter is set to '1', the 'SignalGroup' will only get added to buffer (logged) if one of the signals in the group changes more than its 'DeltaValue'.
Routing	The actual signal that we want to log.
Description	Textual description of the signal to log.

## 4.16. Signals

The following signals are in the SQL Logger component:

Signal Name	Description
LogControl	Start/stop logging (also sets log frequency if enabled in XML).
BufferedEvents	The number of samples in the event buffer.
BufferedSignals	The number of samples in the signal buffer.
BufferedQueries	The number of query requests in the query buffer.
BufferedSQLCommands	The number of SQL commands in the SQL buffer.
EventTableRows	The number of samples in the event table.
SignalTableRows	The number of samples in the signal table.

## 4.17. Alarms

The following alarms are in the SQL Logger component:

Parameter Name	Description
EventTableOverflow	The event table reached the maximum number of rows.
SignalTableOverflow	The signal table reached the maximum number of rows.

## 4.18. Parameters

The following parameters are in the SQL Logger component:

Parameter Name	Description
EnableLogControl	When set, this parameter enables the LogControl signal. Thus, disabling control by message.
EnableDeltaLogging	Enable logging based on change of value.

# 5. Demo Applications

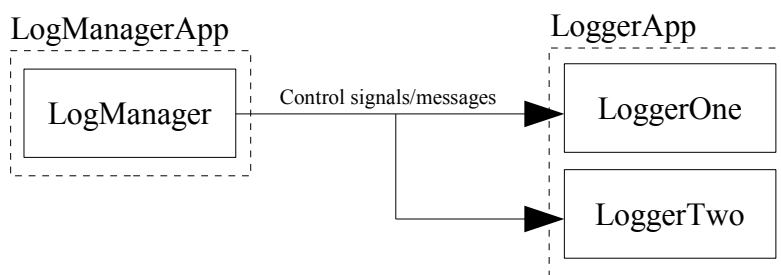
## 5.1. About

This chapter describes how to use the provided demo applications.

## 5.2. Overview

### 5.2.1. Application overview

The demo consists of two applications communicating as shown in the below figure.



### 5.2.2. The LogManager

The LogManager is a simple application for starting and stopping logging in a remote application named LoggerApp. The logging process is controlled either by message or by setting a log control signal. Note that the LoggerApp has to be configured for one of the two by adjusting the XML attribute .

In addition to ordinary log control, the LogManager can send event messages to the LoggerApp.

### 5.2.3. The LoggerApp

The LoggerApp includes two SQL Logger components, each with their own signal group to log. In addition to signal logging, both components are also configured to log incoming event messages. To achieve the best possible performance, each logger has been configured with a separate database file.

### 5.2.4. Demo directory structure

There are four directories within the CDP2SQL demo directory. The Application directory contains the LogManager application files, while the LoggerApp is located in the directory with similar name. The two remaining directories contain the LogManager project files, including source code.

- Application
- CDP\_Application
- LoggerApp
- LogManagerLib

## 5.3. Using the LogManager to control the LoggerApp

### 5.3.1. Getting started

Locate the CDP2SQL demo directory and start the LoggerApp application. When the logger is running, go to the Application directory and start the LogManager application. Now, open a CDPBrowser (or a web browser) and continue with the following sections.

### 5.3.2. Log control by signal

The LogManager controls the signal groups in the LoggerApp by the means of two output signals, `LogControlGroupOne` and `LogControlGroupTwo`. When the signals are adjusted above zero, logging is initiated in the LoggerApp. Setting the signals back to '0', will turn the logging back off.

In some systems it might be useful to log signals based on some external event or an external signal like when a motor is running. The input signals named `MotorOne` and `MotorTwo` has been added to the LogManager to illustrate this concept. When the `MotorOne` input signal is set to '1', logging of signal group one is enabled. If the signal goes back to '0', logging will get disabled again. The input signal named `MotorTwo` controls signal group two in exactly the same way.

#### LogManager Signals

Name	Description	Value	Unit
Process Timer	Process run time each s.	0.000272	s/s
Process Period	Process interval [s].	0.009765	s
LogControlGroupOne	Output signal that is used to control log group one.	1	
LogControlGroupTwo	Output signal that is used to control log group two.	1	
MotorOne	Motor signal that is used to initiate logging.	1 << Log group 1	
MotorTwo	Motor signal that is used to initiate logging.	1 << Log group 2	

#### LoggerOne Signals

Name	Description	Value	Unit
Process Timer	Process run time each s.	3.232e-006	s/s
Process Period	Process interval [s].	1.00004	s
LogControl	Start/stop logging (also sets log frequency if enabled in xml).	1 << Should be '1'	
BufferedEvents	The number of samples in the event buffer.	0	Count
BufferedSignals	The number of samples in the signal buffer.	0	Count
EventTableRows	The number of samples in the event table.	0	Count
SignalTableRows	The number of samples in the signal table.	25 << Should increase	Count
FirstSig	First signal.	0	
SecondSig	Second signal.	0	

#### LoggerTwo Signals

Name	Description	Value	Unit
Process Timer	Process run time each s.	1.679e-006	s/s
Process Period	Process interval [s].	1.00003	s
LogControl	Start/stop logging (also sets log frequency if enabled in xml).	1 << Should be '1'	
BufferedEvents	The number of samples in the event buffer.	0	Count
BufferedSignals	The number of samples in the signal buffer.	0	Count
EventTableRows	The number of samples in the event table.	0	Count
SignalTableRows	The number of samples in the signal table.	20 << Should increase	Count
FirstSig	First signal.	0	
SecondSig	Second signal.	0	

### 5.3.3. Log control by message

In the previous section “Log control by signal”, the control of LoggerApp could have been implemented with messages instead of the `LogControlGroup` signals. The only difference is that a control message would have been sent once, while a signal is continuously sent at the rate requested by LoggerApp. Continue reading for details on how to manually send log control messages from the LogManager.

Before we can send log control messages to the LoggerApp, we have to disable the `LogControl` signal. This is done by setting the `EnableLogControl` parma to '0'. After that we can the send start/stop messages.

**LoggerOne Parameters**

Date, time	Name	Description	Value	Unit
Tue Jan 20 17:29:04 2009	EnableLogControl	When set, this parameter er	0 << Disable signal	
Mon Jan 19 11:33:40 2009	EnableDeltaLogging	Enable logging based on che	0	

**LogManager Messages**

Name	Description	Send
StartLoggingGroupOne	Start remote logging of signal group one.	<input type="button" value="Send"/>
StartLoggingGroupTwo	Start remote logging of signal group two.	<input type="button" value="Send"/>
StopLoggingGroupOne	Stop remote logging of signal group one.	<input type="button" value="Send"/>
StopLoggingGroupTwo	Stop remote logging of signal group two.	<input type="button" value="Send"/>

### 5.3.4. Sending of event messages

The LogManager sends event messages to the LoggerApp by means of the messages named `SendEventGroupOne` and `SendEventGroupTwo`. As the names indicate, the first sends an event message to logging group one (LoggerOne) while the latter sends an event message to logging group two (LoggerTwo). Note that logging has to be enabled in the log group that is going to receive the event. This is done either by setting the control signals or by sending start messages as described in the previous sections.

**LogManager Messages**

Name	Description	Send
SendEventGroupOne	Send an event to logging group one..	<input type="button" value="Send"/>
SendEventGroupTwo	Send an event to logging group two..	<input type="button" value="Send"/>

**LoggerOne Signals**

Name	Description	Value	Unit
Process Timer	Process run time each s.	1.463e-006	s/s
Process Period	Process interval [s].	1.00005	s
LogControl	Start/stop logging (also sets log frequency if e	1	
BufferedEvents	The number of samples in the event buffer.	0 << increases if logging is disabled	
BufferedSignals	The number of samples in the signal buffer.	0	Count
EventTableRows	The number of samples in the event table.	16 << should increase	Count
SignalTableRows	The number of samples in the signal table.	999	Count
FirstSig	First signal.	0	
SecondSig	Second signal.	0	

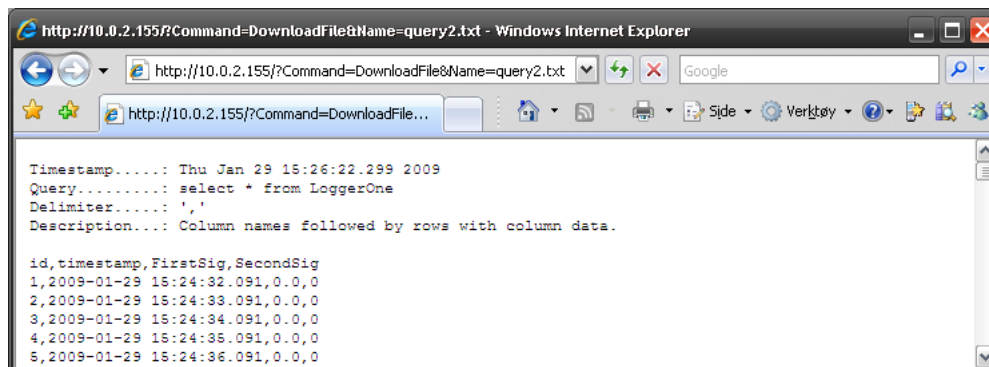
### 5.3.5. Sending of query messages

The LogManager sends a query message to the LoggerApp by means of the message named `SendQueryGroupOne`. In addition to the actual SQL query “`select * from LoggerOne`”, the message contains a delimiter “`,`” and an output filename “`query2.txt`”. To get the query data, log into the LoggerApp web server by using the following command (replace IP with the IP of your LoggerApp):

```
http://10.0.2.155/?Command=DownloadFile&Name=query2.txt
```

**LogManager Messages**

Name	Description	Send
SendQueryGroupOne	Send a query to logging group one.	<input type="button" value="Send"/>



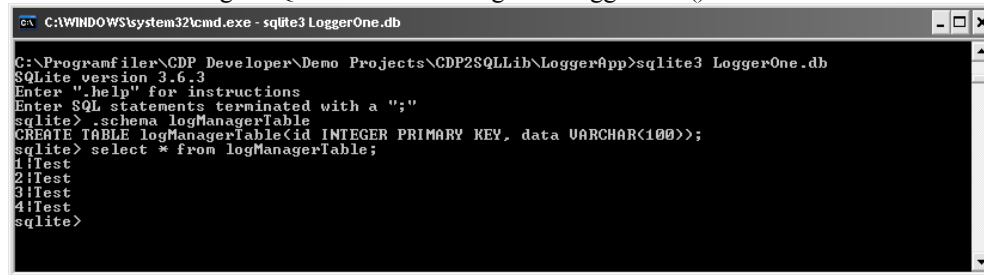
### 5.3.6. Sending of SQL Command messages

The LogManager sends a command message to the LoggerApp by means of the message named SendSQLCommandGroupOne. The first message that is sent will create a table named logManagerTable. After that, messages will insert data in the newly created table. Note that logging has to be enabled for data to get added to the database. This is done either by setting the control signals or by sending start messages as described in the previous sections.

#### LogManager Messages

Name	Description	Send
SendSQLCommandGroupOne	Send a sql command to logging group one.	<input type="button" value="Send"/>

The result of sending 5 SQL command messages to LoggerOne ():



```

C:\WINDOWS\system32\cmd.exe - sqlite3 LoggerOne.db
C:\Programfiler\CDP Developer\Demo Projects\CDP2SQLLib\LoggerApp>sqlite3 LoggerOne.db
SQLite version 3.6.3
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema logManagerTable
CREATE TABLE logManagerTable(id INTEGER PRIMARY KEY, data VARCHAR(100));
sqlite> select * from logManagerTable;
1|Test
2|Test
3|Test
4|Test
sqlite>
  
```

### 5.3.7. Controlling the log frequency

Configuring the LoggingApp with the signal logging attribute LogControlFs set to '1', enables the LogControl signal to control both start/stop and log frequency. The LogManager control signals gets set to '1'. Thus, the logging frequency of the LoggerApp is set 1Hz no matter the value of LogFs.

# 6. Data Extraction

There are myriads of tools for accessing SQL databases in general and SQLite in particular. From simple database browsers, RDBMS configuration managers to scripting languages with SQL bindings.

## 6.1. Command-line database clients

SQLite3 comes with a command-line client for connecting to a database, allowing you to execute SQL commands and queries directly, as well as inspect. From the command-line, be it a terminal window or MS-DOS prompt, run "**sqlite3**" giving the SQLite database filename as an optional argument and type "**.help**" for further details on usage.

## 6.2. Database GUI browsers

SQLite Database Browser is a simple, cross-platform GUI application for accessing the contents of a SQLite database. Supports data modification and exporting to CSV format (Comma-Separated Values) for importing data into other applications. See <http://sqlitebrowser.sourceforge.net/> for details.

For fans of Firefox there exists an add-on called 'SQLite Manager' which is quite feature rich. Supports exporting to CSV and XML. In Firefox click 'Tools->Addons' and search for 'SQLite Manager' to install, or visit <http://code.google.com/p/sqlite-manager/>

## 6.3. Scripting language bindings

Most scripting languages include various database bindings, both generic database-agnostic as well as more database-specific APIs. Thus you can easily access the SQLite database from Perl, Python, Ruby, etc.

### 6.3.1. Python and SQLite3 example code

```
import sqlite3
db = sqlite3.connect('cdp2sql.db')
for row in db.execute("SELECT * FROM SQLLogger"):
    print row
db.close()
```

See <http://www.pysqlite.org/> for details.

### 6.3.2. Ruby and SQLite3 example code

```
require 'sqlite3'
db = SQLite3::Database.new( "cdp2sql.db" )
db.execute( "SELECT * FROM SQLLogger" ) do |row|
  p row
end
db.close
```

See <http://sqlite-ruby.rubyforge.org/sqlite3/faq.html> for details.

# 7. Appendix

## 7.1. Example Component File

```

<?xml version="1.0" encoding="iso-8859-1"?>
<Component Name="SQLLogger" Model="SQLLogger">
  <Activate>1</Activate>
  <fs>10</fs>
  <InitialState></InitialState>

  <Description><![CDATA[
    Component for logging CDP signals/events to a SQL database.
  ]]></Description>

  <!-- Database Configuration -->
  <Database DBName="cdp2sql.db" Type="sqlite3"></Database>

  <!-- Logger Configuration -->
  <LogControl TimeFormat="0" TransactionRetry="5">

    <SignalLogging LogFs="0.2" LogControlFs="1">
      <BufferOptions BufferSize="1000" BufferKeep="100" BufferFlush="40"></BufferOptions>
      <TableOptions MaxRows="999" OverflowMethod="2" OverwriteTable="1"></TableOptions>
      <BurstOptions BurstFs="10" BurstPeriodMs="2000"></BurstOptions>
    </SignalLogging>

    <EventLogging>
      <BufferOptions BufferSize="100" BufferKeep="100" BufferFlush="40"></BufferOptions>
      <TableOptions MaxRows="999" OverflowMethod="2" OverwriteTable="1"></TableOptions>
    </EventLogging>

    <QueryRequest>
      <BufferOptions BufferSize="3"></BufferOptions>
      <FileOptions OutputFile="query.txt" Delimiter=","></FileOptions>
    </QueryRequest>

    <SQLCommand>
      <BufferOptions BufferSize="100" BufferKeep="100" BufferFlush="3"></BufferOptions>
    </SQLCommand>
  </LogControl>

  <!-- Information Table -->
  <InfoTable Overwrite="0" ForceName="">
    <Line Name="Date" Value=""></Line>
    <Line Name="Time" Value=""></Line>
    <Line Name="Logging" Value="MANUAL"></Line>
    <Line Name="IMO Number" Value="IMO9181065"></Line>
    <Line Name="Unit Number" Value="xxxxxxxxxx"></Line>
    <Line Name="Signal Type" Value="VIBRATION"></Line>
    <Line Name="Measuring Time" Value="5"></Line>
    <Line Name="Sampling Freq" Value="30000"></Line>
    <Line Name="Azimuth" Value=""></Line>
    <Line Name="Pitch" Value="80"></Line>
    <Line Name="Load" Value=""></Line>
    <Line Name="Rpm" Value="1500"></Line>
  </InfoTable>

  <!-- The SignalGroup with SignalSource elements representing signals we want to log -->
  <SignalGroup>
    <SignalSource Name="FirstSig" Type="double" DeltaValue="1.0" Routing="Thruster1.Temp"
      Description="First signal."></SignalSource>
    <SignalSource Name="SecondSig" Type="int" DeltaValue="1" Routing="Thruster2.Temp"
      Description="Second signal."></SignalSource>
  </SignalGroup>

  <!-- Component signals -->
  <Signals>
    <Signal Name="LogControl" Input="1" Type="double" Unit="" Value="" Routing="No routing"
      Description="Start/stop logging (also sets log frequency if enabled in xml)."></Signal>
    <Signal Name="BufferedEvents" Input="0" Type="int" Unit="Count"
      Description="The number of samples in the event buffer."></Signal>
    <Signal Name="BufferedSignals" Input="0" Type="int" Unit="Count"
      Description="The number of samples in the signal buffer."></Signal>
  </Signals>

```

```
<Signal Name="BufferedQueries" Input="0" Type="int" Unit="Count"
  Description="The number of query requests in the query buffer."></Signal>
<Signal Name="BufferedSQLCommands" Input="0" Type="int" Unit="Count"
  Description="The number of SQL commands in the SQL buffer."></Signal>
<Signal Name="EventTableRows" Input="0" Type="int" Unit="Count"
  Description="The number of samples in the event table."></Signal>
<Signal Name="SignalTableRows" Input="0" Type="int" Unit="Count"
  Description="The number of samples in the signal table."></Signal>
</Signals>

<Alarms>
  <Alarm Name="EventTableOverflow" Group="" Level="Warning" Trig="0" Enabled="1" EnabledState=""
    Signal="" Inverted="0" SignalOutSet="" Text="The event table reached the maximum number of rows."
    Description="The event table reached the maximum number of rows."></Alarm>
  <Alarm Name="SignalTableOverflow" Group="" Level="Warning" Trig="0" Enabled="1" EnabledState=""
    Signal="" Inverted="0" SignalOutSet="" Text="The signal table reached the maximum number of rows."
    Description="The signal table reached the maximum number of rows."></Alarm>
</Alarms>

<Parameters>
  <Parma Name="EnableLogControl" Unit="" Value="1" Min="0" Max="1" DefaultValue="1" PreviousValue="0"
    TimeLastChanged="Thu Dec 04 12:29:26 2008" Description="When set, this parameter enables the
    LogControl signal. Thus, disabling control by message."></Parma>
  <Parma Name="EnableDeltaLogging" Unit="" Value="0" Min="0" Max="1" DefaultValue="0" PreviousValue="1"
    TimeLastChanged="Mon Jan 19 12:07:23 2009" Description="Enable logging based on change of value.">
  </Parma>
</Parameters>

<Subcomponents>
</Subcomponents>

<RemoteComponents>
</RemoteComponents>
</Component>
```