Implementation and Testing of RDB Archiver with MySQL

Ruizhe Ma, DePauw University

Supervisor: Dr. Richard Farnsworth, Argonne National Laboratory

Introduction

The Relational Database Channel Archiver (RDB Archiver) archives operational data of an accelerator. Developed by Kay Kasemir at Oak Ridge National Laboratory, RDB Archiver collects data from Input/output Controllers (IOCs) and writes into a Relational Database. The project involves implementing and testing the archiver with MySQL database and serves as a foundation for future developments and the potential application of RDB Archiver at the Advanced Photon Source (APS) of Argonne National Laboratory. The first part of the paper describes detailed steps of the setup of RDB Archiver with MySQL and related configurations for testing. The second part demonstrates the performance of the archiver with MySQL database.

Overview

RDB Channel Archiver is developed as one part of the Control System Studio (CSS). Built as Java/Eclipse Rich Client Platform (RCP) products, CSS is a collection of tools including the Channel Archiver, the Data Browser and other control system diagnostic tools and operator interfaces. An Archive Engine samples Process Variable (PV) data, i.e. named control system data points that have a value, time stamp and so on, from EPICS (Experimental Physics and Industrial Control System[1]) IOCs via Channel Access (CA, the EPICS network protocol), and places them in a Relational Database that has required table structure. Configurations of the Archive Engine are also stored in RDB. Users can then access historic data from the database as well as live data using the CSS Data Browser. See Figure 1, “RDB Archive System Architecture”.

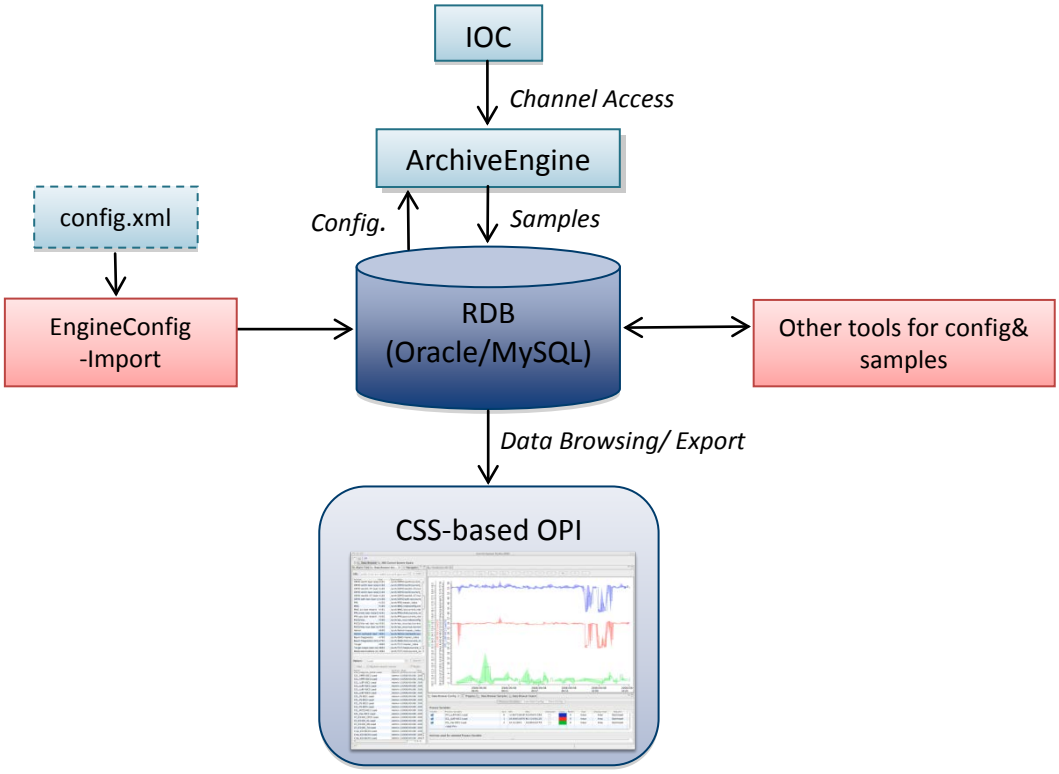


Figure 1.RDB Archive System Architecture

Building the Archive Engine

The Archive Engine is a standalone Java/Eclipse RCP product. It uses a selected set of Java libraries called Plug-ins (the *.JAR files) to fulfill its functionality. All the Java sources for the Archive Engine built in this test are downloaded from the SNS CSS web page [2]. However, for future developments it is suggested to use the latest sources from the Source Forge repository using the Mercurial version control system [3].

In addition to the sources, you will need

- Java JDK. This should be the Sun/Oracle or Apple JDK, not just a JRE. You also need to set the related system environment variables for Java after installation.
- Eclipse IDE for RCP and RAP Developers [4]. Using other versions of Eclipse may result in missing required plug-ins that are part of Eclipse.

When start the Eclipse IDE for the first time, you will be prompt for setting a `Workspace`. This is the directory where Eclipse keeps configuration parameters. The Workplace is not necessarily the same directory where the source files are, but no matter what directory you choose, you will have to "import" your source files into the workplace to make them recognizable to Eclipse. Select your desired directory, or use the default directory.

To import the sources into workspace:

- In Eclipse, select `File->Import … General->Existing Projects into Workspace`
- As a root directory, select the directory where the source files are, for example `C:/css_souces`
- Do not check the option "Copy projects into workspace"
- By default all the plug-ins in the directory will be selected. You need to select specific plug-ins for your product. For the Archive Engine select

  - org.csstudio.apputil
  - org.csstudio.archive.engine
  - org.csstudio.archive.rdb
  - org.csstudio.auth
  - org.csstudio.data
  - org.csstudio.java
  - org.csstudio.logging
  - org.csstudio.platform.libs.epics
  - org.csstudio.platform.libs.jdbc
  - org.csstudio.platform.libs.jms
  - org.csstudio.platform.utility.rdb
  - org.csstudio.utility.httpd
  - org.csstudio.utility.pv
  - org.csstudio.utility.pv.epics
  - org.csstudio.utility.simu*
  - org.csstudio.workspace

  *This plug-in generates simulated signals and is not required by the Archive Engine per se, but experimenting with simulated signals is a convenient way to verify whether your Archive Engine works and you can also test your Archive Engine's performance with simulated signals.
- Press `Finish`

This list should include all the plug-ins that the Archive Engine needs to function. In cases that errors exist about missing plug-in dependencies, first make sure that you are using the right version of Eclipse IDE as described above. Then, locate the plug-ins with a red error mark in the Eclipse Project Explorer on the left side of Eclipse window by default, and expand the plugin directory tree. In the META-INF folder there is a file named `MANIFEST.MF` that describes plugin dependencies, i.e. which other plugins are required to load this plugin. Open the `MANIFEST.MF` file in Eclipse and go to the "`Dependencies`" tab; the missing plugin(s) would be denoted with a red error mark. See Figure 2, "Finding Missing Plug-ins".

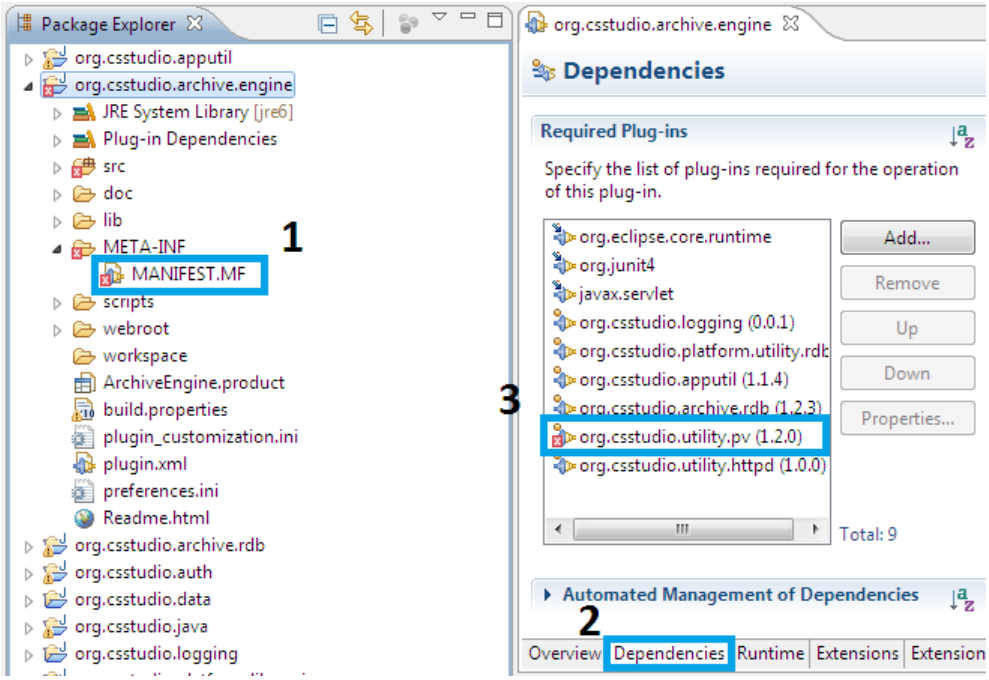


Figure 2. Finding Missing Plug-ins

After follow the steps described above and import the missing plugins, there should be no errors. This is actually the trial and error way of how to find all the necessary plugins for a product that you are not familiar with. First import the source file that contains the `*.product` file, look for missing plugin dependencies in the `MANIFEST.MF` file, and then import the missing plugins.

Next, open the `*.product` file of the product you are building. For the Archive Engine, open

```
org.csstudio.archive.engine/ArchiveEngine.product
```

The Eclipse Product Editor will open. See Figure 3, "Eclipse Product Editor". During the development and initial testing, it is more convenient to execute products from within Eclipse.

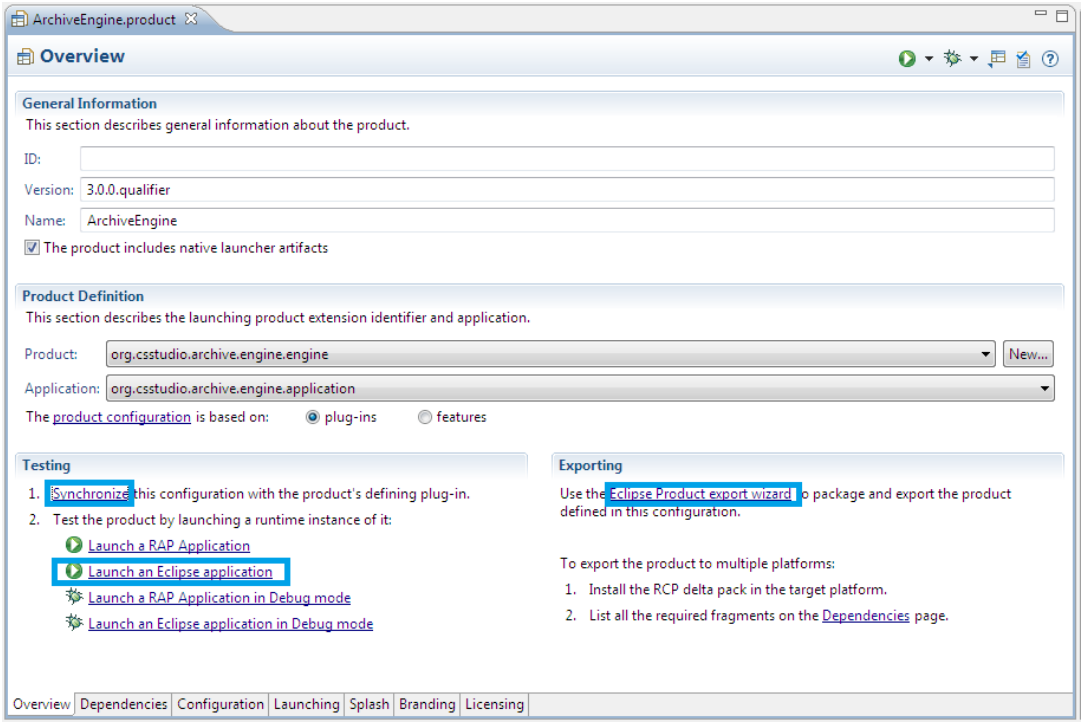


Figure 3. Eclipse Product Editor

In the `Overview` tab, press `Synchronize`, then `Launch an Eclipse application` to run the product from within Eclipse. The product should start.

In the case of the Archive Engine, it should soon exist with an error message indicating that it requires command-line arguments. This is fine for now because when running a product from within Eclipse for the first time, the main goal is that the product should start. It may then stop because of missing command line arguments, but there should be no errors regarding missing plugins or compilation problems. To add command-line arguments to products executed in Eclipse, go to menu `Run->Run Configurations … Arguments` tab. Add your desired command-line argument in the "Program Arguments" section, separated by space.

However, if there are no other errors found except missing command-line arguments and you have some faith in the Archive Engine, you can proceed to export the product and test the command-line arguments later with the real product.

To export the product, press `Eclipse Product export wizard` in the Eclipse Product Editor.

- As a root directory, enter the name of the product (Archive Engine in this case)
- In the destination section, enter a directory where you wish to place the exported product, for example `C:/CSS`
- You might want to uncheck the option to "Generate metadata repository"
- Press Finish

Now you should have a directory like `C:/CSS/Archive Engine` that contains an ArchiveEngine executable. You can copy that directory to elsewhere and run the Archive Engine there.

The Archive Engine is a command-line application. Such applications, in Eclipse called "headless" RCP applications, are invoked from a terminal window, i.e. the Linux shell, Mac OS X terminal, or Windows Command Prompt. They are configured via command-line arguments, and they print information to the terminal. **However, there is currently a limitation for command-line products on Windows**. In the Windows command line tool `cmd.exe` you will not see any output from headless RCP applications by default. For the time being, the only solution is to manually replace the generated launcher, such as `ArchiveEngine.exe`, with a copy of `eclipsec.exe` from the root directory of Eclipse IDE, and rename `eclipsec.exe` to your desired name, for example ArchiveEngine.exe. Now you can invoke the product from within the Windows Command Prompt window.

Note:

The product that you export from Eclipse is by default limited to the operating system on which it was exported. In this project, the Archive Engine and other products are built and run on the Window 32-bit OS. To export code for different platforms, you will need the Eclipse "Delta Pack", or simply build and export the product in the operating system that you want the product to run on. For more details about cross-platform export, consult the Control System Studio Guide [5], section 3.5.

Building Other Products

The `ArchiveConfigTool` is a tool used to export engine configuration into a XML file or import the configuration files into the relational database for new engines. It is defined in the product file:

```
org.csstudio.archive(.config).rdb/ArchiveConfigTool.product
```

You will be able to build the tool following the same steps of building the Archive Engine. It must be noted that the product built from the sources downloaded from the SNS CSS source snapshot is incomplete; the "-import" command is broken. To fix the problem, use the latest sources from the Source Forge repository. In this project, engine configuration is done by directly updating the table content in the MySQL database using MySQL commands; thus this tool is built but not used.

The Control System Studio is used mainly for data browsing in this project. It is very confusing at first glance because while the whole collection of tools, including the `ArchiveEngine`, the

`ArchiveConfigTool` and the `AlarmServer`, is called Control System Studio, there is also a product that contains the `Data Browser`, `PV table` and other tools named Control System Studio, too. Here is a simple way to understand this: think of "Control System Studio" as the name of the collection of plugins; while all the tools are built using these plugins, some tools can be separate programs like the `ArchiveEngine` and the `ArchiveConfigTool`, and some others can be integrated into a program that has a Graphical User Interface (in this case, the CSS product).

To build the CSS product, located the product file:

```
org.csstudio.sns.product/SNS_CSS.product
```

Because in this project only the Data Browser is used, CSS is not built from the sources; instead, the compiled SNS version of CSS is downloaded from the SNS CSS webpage and used with required setting changes from within the program to suit for testing. In the future, it would be better to develop an APS version of CSS with all the related tools embedded.

Relational Database Setup and Engine Configuration

Several CSS tools interface to a relational database (RDB). The Archive Engine reads configuration from an RDB and writes data into it, and the Data Browser retrieves history data from an RDB. Therefore, basic RDB administration skills are needed for CSS administrators. Currently, MySQL, Oracle and PostgreSQL are supported. Users can compare the tradeoffs between different databases and choose according to site needs. This project focuses on the performance of MySQL.

If you do not have a MySQL server on hand, or you want to test it before moving to the real database of your site, download the "Community Edition" of MySQL [6] and install it on your computer per MySQL online documentation [7]. MySQL online documentation is also a good place to start with if you have no experience with MySQL.

In principle, three types of RDB users are needed:

- Administrator account: A user that has all privileges to the database.
- Write-access user: A user that has "write" privilege to the database. This user account is used by the Archive Engine to write data into the database.
- Read-only user: A user that only has "read" privilege to the database. This user account is used by the CSS Data Browser and other end users to access the archived data without being able to make any changes to them.

In the case of testing, we make no distinctions between these three types of accounts as in this project.

To access the RDB, you have to provide a URL of MySQL to the CSS tools in the following format:

```
jdbc:mysql://[host]:[port]/[database]
```

You will also need to provide a username and the associated password for the RDB. End users are typically given a read-only account, and the Archive Engine a write-access account.

Before running the Archive Engine, you need to create the required table structure in MySQL. The configuration of engine is also stored in MySQL as table entries. Locate the file:

```
org.csstudio.archiver.rdb/dbd/mysql_schema.txt
```

This schema documents the commands to create the table structure for MySQL. In reality, there might be minor changes in the structure according to specific site needs, but this is the basic table structure that the Archive Engine needs to store data in MySQL. See Figure 4, “MySQL Model”.

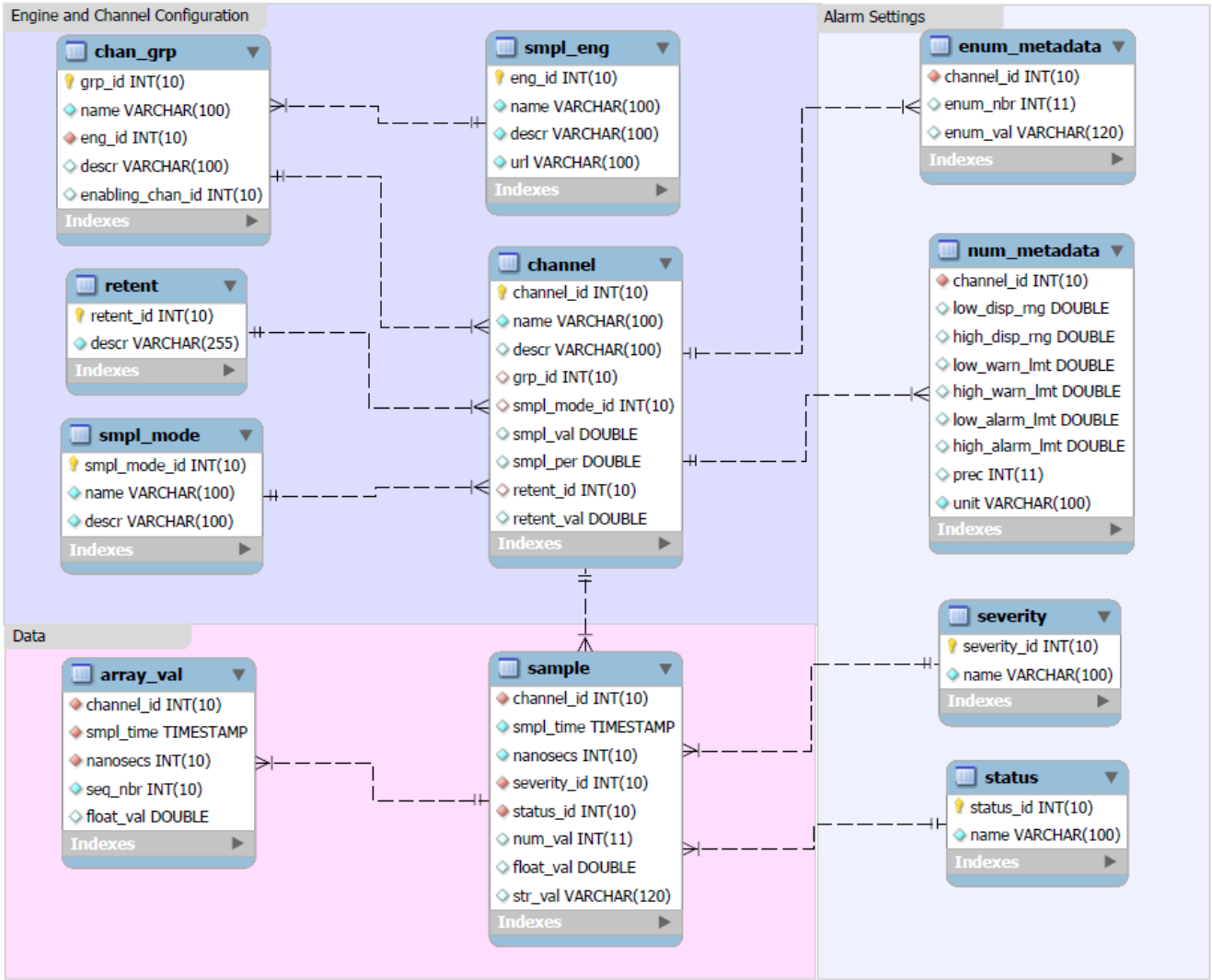


Figure 4. MySQL Model

Engine and Channel Configuration

The Archive Engines are configured in these tables. You can manually insert configurations into the tables, or import the configuration files (`*.xml`) of the old engines into the RDB using the `ArchiveConfigTool`.

- smpl_eng
  This table stores the configuration of the Archive Engines. For example,

eng_id	*name*	*descr*	*url*
1	Power	Power Supply System	http://localhost:4812
2	Water	Water Supply System	http://localhost:4813

  The *eng_id* is a unique integer to identify an engine. The engine's *URL* is where the engine runs. The port numbers are unique for each engine but can be arbitrarily chosen. This url will be needed to view the engine's status.

- chan_grp
  This table stores the configuration about the group organization. For example,

grp_id	*name*	*eng_id*	*descr*	*enabling_chan_id*
1	401-P	1	Power Supply in Building 401	Null
2	Lodging -P	1	Power Supply in Lodging	Null
3	401-W	2	Water Supply in Building 401	Null
4	Lodging -W	2	Water Supply in Lodging	Null

  The *grp_id* is a unique integer to identify a channel group. The *eng_id* defines the ID of the engine that this group belongs to. The *enabling_chan_id* defines the ID of the channel which controls the archiving status of all the channels in this group. Users can enable or disable the entire archive group based on the status of this particular channel.

  The separation of channels into groups is meant for the engineers who maintain the sample engine configuration to group the channels as needed, such as by location, by associated front-end computer, and by functionality.

- channel
  This table stores the configuration of the channels (PVs). For example,

channel_id	*name*	*descr*	*grp_id*
1	401:P:Bsmt	Power in 401 Basement	1
2	401:P:1	Power in 401 1st Floor	1
3	401:W	Water Supply in 401	3

smpl_mode_id	*smpl_val*	*smpl_per*	*retent_id*	*retent_val*
1	Null	Null	9999	Null
1	Null	Null	9999	Null
2	0.1	60	9999	Null

The *channel_id* is a unique integer to identify a channel. The channel's *name* has to be a valid PV name. The *grp_id* defines the group that the channel belongs to. The *smpl_mode_id* defines the sample mode of the channel. The *smpl_val* defines the threshold value for the value change threshold filter. See *smpl_mode* for details on sample modes. The *smpl_per* defines the sample rate in seconds. The *retent_id* saves information regarding the length that the user wishes to save the data. It is used for archived data management and manipulation. The *retent_val* has no meanings for now.

This is a hierarchical organization: each sample engine has one or more groups, and each group has one or more channels.

- smpl_mode

  This table stores the metadata about sample modes. For example,

smpl_mode_id	*name*	*descr*
1	Monitor	Store every received update
2	Scan	Periodic scan

The Archive Engine supports several sample modes. Since there are essentially infinite data that can be stored but the resources (CPU power, disk space, network bandwidth) are finite, it is better to store fewer samples according to needs.

- Monitored

In this mode, each received sample is written to the disk. With proper configuration at the IOCs, i.e. the IOCs only pass significant changes to the Archive Engine, this mode is ideal and highly recommended.

-Monitored With Threshold

This mode is a variation of the Monitored mode, adding a value change threshold filter. When the channel is configured in the Monitored mode with *smpl_val* greater than 0, this mode is active, writing only the samples that differ from the last written sample by at least the value of *smpl_val*. Ideally, such filtering process is performed by the front-end computers to reduce network traffic, but if this is impossible, the engine can do the work in the price of CPU load.

-Scanned

In this mode, the Archive Engine still receives each update from the data source, but it only writes the most recent sample at periodic times, specified by *smpl_per*.

This mode is designed for channels which do not have a good dead-band configuration, where using the monitored mode would add too many samples to the database. Periodic sampling is imperfect in the fact that it might miss important data points during standby periods, but it is a workable compromise in some occasions.

- retent

  This table stores the metadata about retention. For example,

retent_id	*descr*
9999	Forever
30	A month

Alarm Settings

All the metadata, status and severity are configured in EPICS IOCs. The Archive Engine writes this information into the tables when it fetches samples from the IOCs.

- enum_metadata
  This table stores the metadata of channel status for enumerated channels.

- num_metadata
  This table stores the metadata of channel status for numeric channels. For example,

channel_id	*low_disp_rng*	*high_disp_rng*	*low_warn_lmt*	*high_warn_lmt*
1	0	10	3	7

low_alarm_lmt	*high_alarm_lmt*	*prec*	*unit*
1	9	3	mA

  This configuration means: channel 1 sends out signals that range from 0 mA to 10 mA; warnings are sent when the signal falls between 1 mA and 3 mA or between 7 mA to 9 mA; and alarms are set off when the signal falls below 1 mA or above 9 mA. “prec”, precision, is used for displaying number. In many CSS tools, such as the Data Browser, you can choose to use different precision settings rather than the one that you get from the metadata.

- severity
  This table stores information about severity settings. For example,

severity_id	*name*
1	OK
2	MINOR
3	MAJOR
4	INVALID

  Using the num_metadata configuration above, a severity of “OK” corresponds to a signal between 3 mA and 7 mA; a “MINOR” corresponds to the warning level; a “MAJOR” correspond to the alarm level; and an “INVALID” correspond to out of range values.

- status
  This tables stores information about the status of the channel. For example,

status_id	*name*
1	OK
2	Disconnected
3	High
4	Way High
5	Low
6	Way Low
7	Archive_Off

  The status information is similar to the severity information of the signal, but is more specific. “High” and “Low” correspond to a “MINOR” severity, and “Way High” and “Way Low” correspond to a “MAJOR” severity.

Data

Samples are stored in these two tables.

- sample
  Numeric data are stored in this table along with their time stamp. Different from the table structure for Oracle, for MySQL there is a separate column for nanoseconds.

channel_id	*smpl_time*	*nanosecs*	*serevirty_id*	*status_id*
1	2011-07-30 12:22:02	312000000	2	5

num_val	*float_val*	*str_val*
Null	2.0610737385376505	Null

- array_val
  For double arrays, the *float_val* in the sample table contains the first array element, and the array_val table has the following ones.

Preference Settings

Each plugin with configurable settings should have a file `preference.ini` in its root directory. This file provides the default settings for the plugin. The Eclipse preference service for a plugin will automatically use such a file as long as it has the correct name and is located in the plugin’s root directory.

When building site-specific products, the file `plugin_customization.ini` in the plugin that defines that product is automatically used by Eclipse to override settings from individual plugins. All settings in this global preference file are prefixed with the name of the affected plugin, for example, to configure the schema prefix of the RDB you should write:

```
org.csstudio.archive.reader.rdb/schema=
```

If you wonder about the names of preference settings supported by each plugin, consult the `preferences.ini` of the respective plugin.

In the end, the preferences configuration is performed before compile the product. However, for testing it is most convenient to save a copy of the preference file, for example, `test.ini`, and run the product with a command-line option:

```
archiveengine.exe –pluginCustomization /path/to/test.ini  -…
```

If you save the `test.ini` in the root directory of the product, you do not need the path to the file, for example using

```
–pluginCustomization test.ini
```

For testing the Archive Engine, you want to write the following entries into `test.ini` including preferences as well as documentations, for example:

```
# Engine Preferences for testing at APS
# @author Ruizhe Ma

# Archive RDB Access
# MySQL address (APS)
org.csstudio.archive.rdb/url=jdbc:mysql://164.54.8.243:3306/archive

# User name and password with write-access
org.csstudio.archive.rdb/user=ruizhema
org.csstudio.archive.rdb/password=ruizhema

# Schema for MySQL (leave blank for MySQL)
org.csstudio.archive.reader.rdb/schema=

# Channel Access address for sampling from IOCs
org.csstudio.platform.libs.epics/use_pure_java=false
org.csstudio.platform.libs.epics/addr_list=127.0.0.1
org.csstudio.platform.libs.epics/auto_addr_list=false

# Seconds between log messages for Not-a-Number, futuristic, back-in-time values, buffer
#overruns; 24h = 24*60*60 = 86400 seconds
org.csstudio.archive.engine/log_trouble_samples=86400
org.csstudio.archive.engine/log_overrun=86400

# Write period in seconds
org.csstudio.archive.engine/write_period=30
```

```
# Maximum number of repeat counts for scanned channels
org.csstudio.archive.engine/max_repeats=60

# Write batch size. Instead of performing insertion for each sample, the Archive Engine writes
# a group of samples into the RDB as a batch
org.csstudio.archive.engine/batch_size=500

# Buffer reserve (N times what's ideally needed, i.e. buffer_reserve*(write_period/scan_period))
org.csstudio.archive.engine/buffer_reserve=2.0

# Samples with time stamps this far ahead of the local time are ignored
# 24*60*60 = 86400 seconds = 1 day
org.csstudio.archive.engine/ignored_future=86400
```

Note: the "ignored_future" is hard-coded into the product at ignored_future=86400 in the old version available from the SNS CSS website. In newer versions, the "ignored_future" can be adjusted through preferences.

Running Archive Engines

The Archive Engine is a command-line tool. It supports these command-line arguments:

```
-help                          : Display Help
-port 4812                     : HTTP server port
-rdb_urljdbc: ...              : Database URL, overrides preference setting
-rdb_userarch_user             : Database user, overrides preference setting
-rdb_password secret           : Database password, overrides preference setting
-engine demo_engine            : Engine config name
-pluginCustomization /path/to/mysettings.ini    : Eclipse plugin defaults
-data /home/fred/Workspace : Eclipse workspace location
```

Ideally, the RDB preferences are configured in the `plugin_customization.ini` file or other configuration files such as `test.ini`, so the actual command in testing looks like:

```
archiveengine –engine Demo –pluginCustomization(/path/to/)test.ini
–port 4810(–data Demo)
```

The Archive Engine is an Eclipse product which uses a "Workspace". Compared to the CSS product it hardly uses the workspace. Its only use might be the `.metadata/.log` file that could contain error messages of the running instance. In any case, each Archive Engine instance must have a unique workspace directory, and the suggestion is to use the engine name as the workspace name. The "`-data Demo`" argument will automatically create the directory `Demo` if not existed. The "`-port`" option is required to start the engine's web server and also serves as a

consistency check; the port number must match the number in the URL of the engine's configuration. Without specifying the port number, the Archive Engine will use "`-port 4812`" by default.

When start the Archive Engine, you may get an error message like Figure 5, "Error Message".

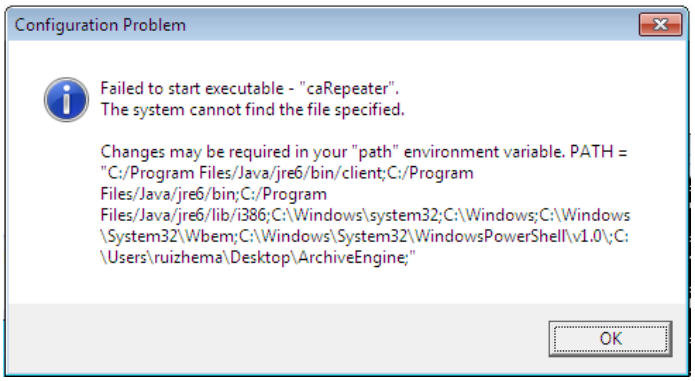


Figure 5. Error Message

This is fine for testing the engine and it will not impact the engine. In real operation, each computer that uses CA needs to run an EPICS Channel Access repeater to automatically re-connect to IOCs after network issues.

The Archive Engine has a built-in web server for status viewing and basic remote control of the engine. When the engine is running, its web server is accessible via

```
http://<engine's URL>/main
```

For example,

```
http://127.0.0.1:4812/main
```

Other URLs of the web server includes:

`http://<engine's URL>/stop`	The engine will write a final Archive_off sample to each channel, then stop.
`http://<engine's URL>/restart`	The engine will restart. Invoking this URL is required after changes to the configuration of an archive engine
`http://<engine's URL>/reset`	Invoke this URL to reset engine statistics displayed on the main page of the engine
`http://<engine's URL>/environment`	Invoke this URL to display engine environment settings which may be useful to debug a problem

These URLs must be entered directly into the web browser. They are not accessible as links from the main page.

Using the Data Browser

End users will use the Data Browser to access archived as well as live data. In this project, the compiled SNS CSS product is used.

To configure the connection between the CSS and your RDB, select `CSS->Preferences…CSS Applications->Trends->Data Browser`. Add your MySQL URL, for example,

```
jdbc:mysql://164.54.8.243:3306/archive
```

in the `Archive Data Server URLs` and the `Default Archive Data Sources` sections. You can remove other SNS URLs in the CSS since you do not need them anyway. You also need to provide a MySQL account, typically a Read-only account for end users, in `Trends->RDB Archive`, and leave other entries empty. See Figure 6, “CSS Configuration”.

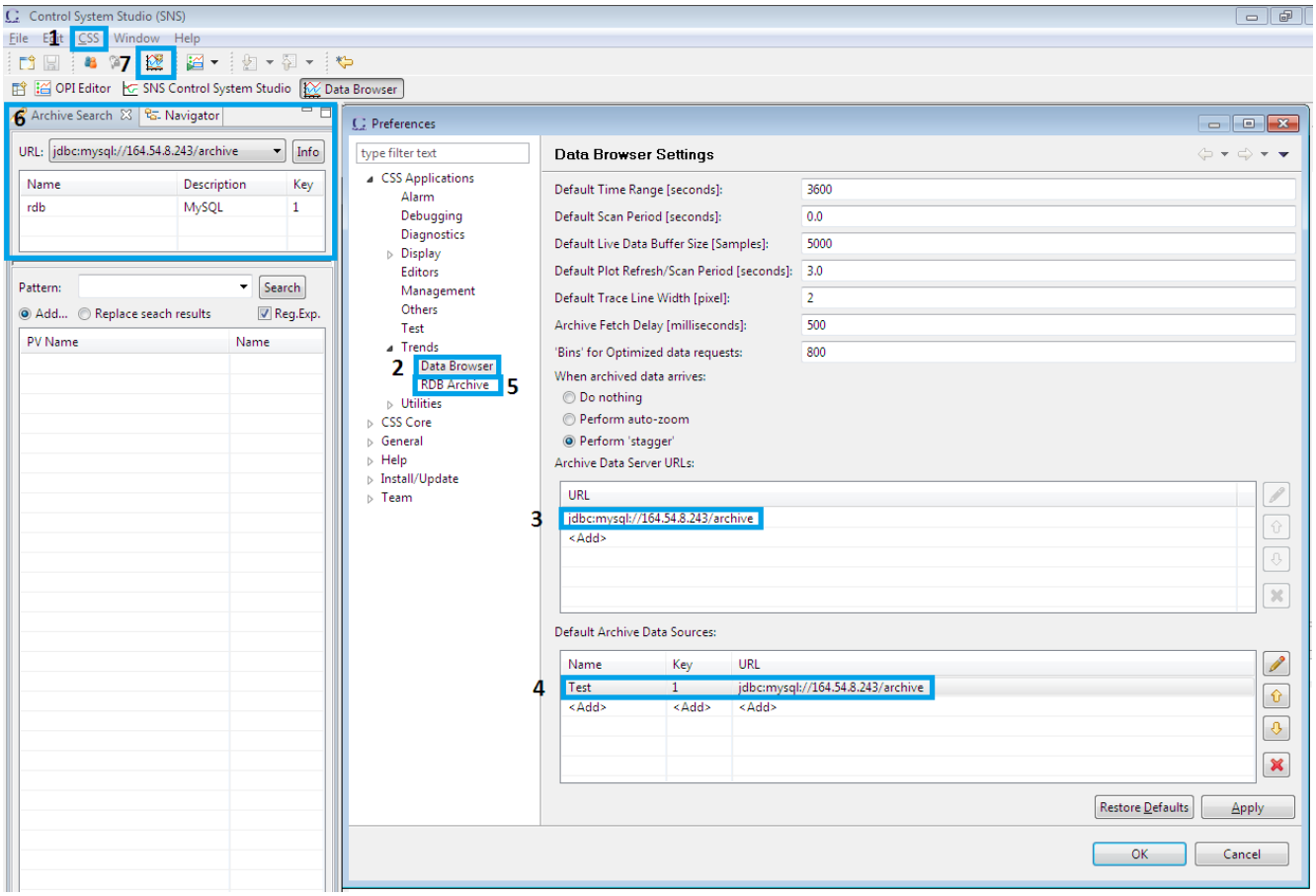


Figure 6. CSS Configuration

Once you see an entry like the one shown in window 6 of Figure 6, you are successfully connected to MySQL. You may need to restart the CSS to make it work.

After connect to MySQL, click the Data Browser icon (window 7) to start the Data Browser. Right click in the Data Browser’s interface, select “Add PV”, and type in a PV name, for

example, “sim://sine”. The Data Browser will then display the plot of “sim://sine” if it is a valid PV name. The archived data for “sim://sine” will also be shown, if any. See Figure 7, “Data Browser”. For more information about how to use the Data Brower, consult the online help of the CSS.

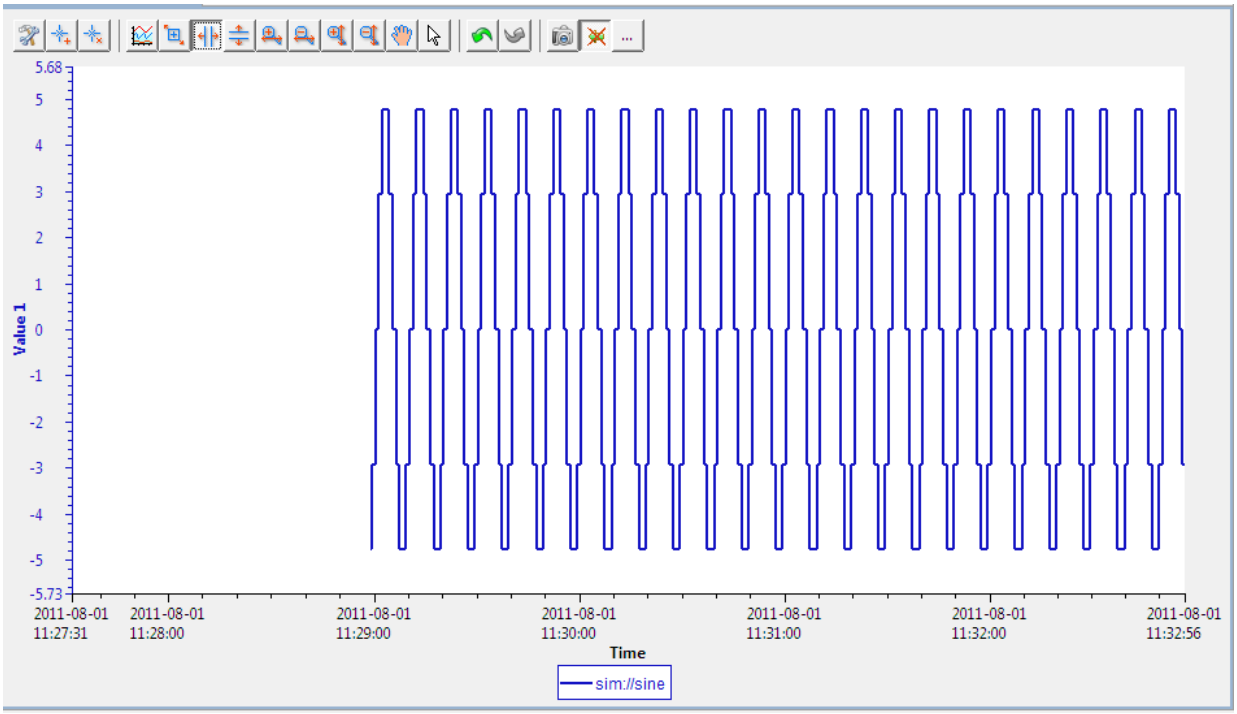


Figure 7. Data Browser

Test Design and Procedure

To test the archive speed of the Archive Engine with MySQL, simulated signals are used. The engine’s plug-in, `org.csstudio.utility.pv.simu`, creates simulated PVs that update periodically, and by dividing the number of archived samples by the run time we are able to calculate the archive speed.

A group of PVs are created with Excel spreadsheets and imported into the RDB manually. The PVs are named in the pattern “sim://sine(0,10,n,0.1)”, meaning that the PVs are sinusoidal waves valued 0 to 10, split into n updates, updating every 0.1 seconds. By controlling the number of active PVs we are able to set the desired sample update speed. For example, a group of 100 such PVs would generate approximately 1000 samples per second. If the archive engine functions normally without data loss, the archive speed would be 1000 samples per second, too. In reality, because the simulated PVs operate on a simple timer, the update period is only an approximation and the sample speed would not be exact 1000 samples per second. An alternative and more precise way to determine if any data are lost is to look at the web interface of the engine. In the

Group display there is a column named “Overruns” that records the number of lost samples. In the experiment, groups of 50, 100, 150, 200, 300, and 500 PVs are tested.

Many factors can potentially impact the archive speed, including the `Write_period`, the `Buffer_reserve`, the `Batch_size`, the network traffic, the storage engine of MySQL, the CPU of the computer where the Archive Engine runs, the memory of the Java Virtual Machine, etc. This project focuses on the effect of the Archive Engine’s parameters that can be configured through the `*.ini` preference file.

In this project, the Archive Engine runs on a Microsoft 32-bit OS with Intel(R) Pentium(R) 4 CPU of 3.20GHz. The database is MySQL, server version 5.0.77 for redhat-linux-gnu (x86_64), using MyISAM storage engine. All the “default settings” of the Archive Engine refers to the settings specified in the `test.ini` on page 13 of this documentation, i.e. `Write_period=30 seconds; Buffer_reserve=2.0; Batch_size=500`. The test runs are 5 minutes long as shown by the `uptime` entry in the web interface of the engine, if not specified otherwise. The engine runs two trials with each set of configuration, and the average value of the archive speed is recorded.

After the channel configurations are uploaded into MySQL, the engine is started from the Windows Command Prompt. Open the main page of the engine’s web page to check the uptime and other status. When time is up, refresh the page `http://<engine’s URL>/stop` to stop the engine. Go back to the command prompt window and wait until it shows that the engine has completely stopped as the writer generally has not finished writing data into the database although the engine is turned off. When the engine stops, check the samples in MySQL via

```
show table status;
```

Rows in the table *Sample* is the count of archived samples. Dividing the number of archived samples by the uptime will give the archive speed. The average sample size is calculated by dividing the size of *Sample* by the number of samples. The size of the table in bytes is estimated by the formula

```
(Data_length+Index_length+9000)/Rows,
```

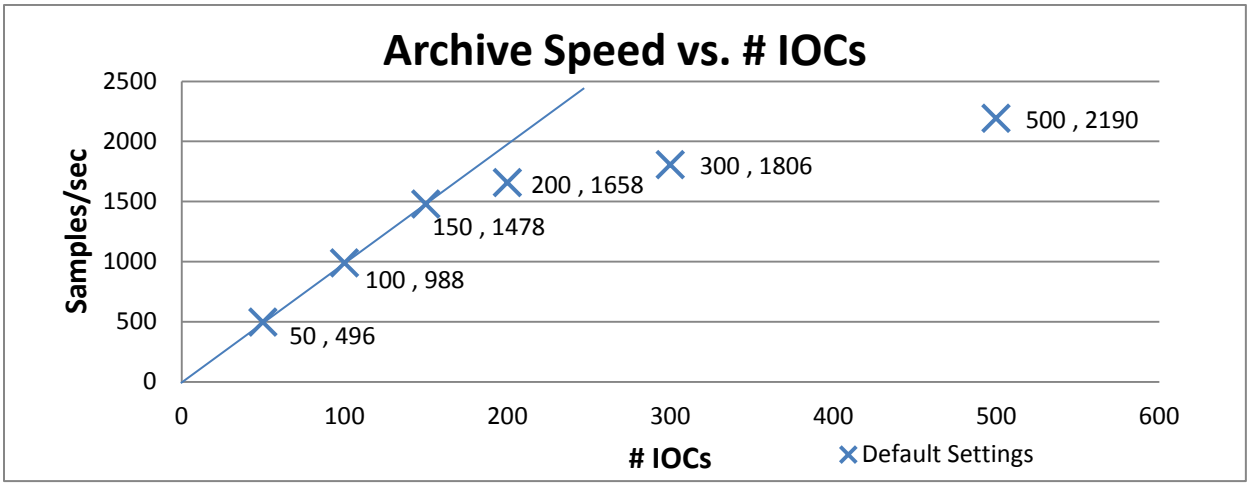
where 9000 represents the size of the `.frm` file that cannot be obtained via commands. After recording all the needed data, erase the samples from *Sample* via

```
delete from sample;
```

and repeat the steps for the next trial.

Test Results and Analyses

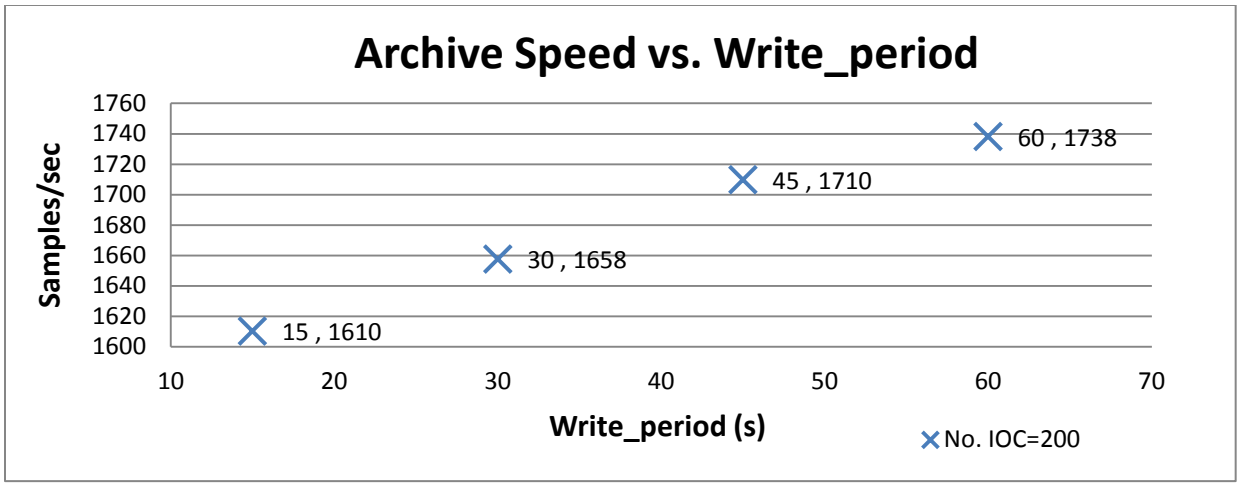
The first series of tests are conducted with the default settings. The archive speed is expected to grow with the sample update speed controlled by the number of PVs until the engine reaches its limit when the archive speed ceases to increase. Graph 1 is the plot of the archive speed against the number of IOCs, i.e. the number of PVs.



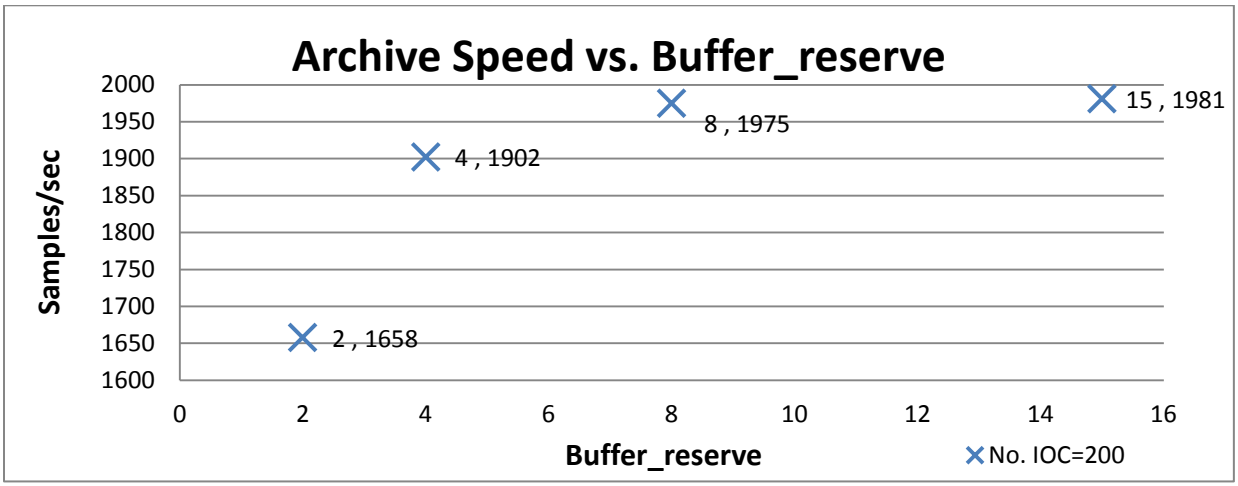
Graph 1. Archive Speed vs. # IOCs

The archive speed of the engine grows linearly with the number of IOCs and is close to the designed value of 500, 1000 and 1500 samples per second when the number of IOCs is 50, 100 and 150, respectively, indicating that the engine has not yet reached its capacity. However, as the number of IOCs grows beyond 200, the growth of the archive speed slows down and the archive speed no longer matches the sample update speed, meaning data are lost. With this setting, the maximum archive speed of the engine is around 1500 samples per second.

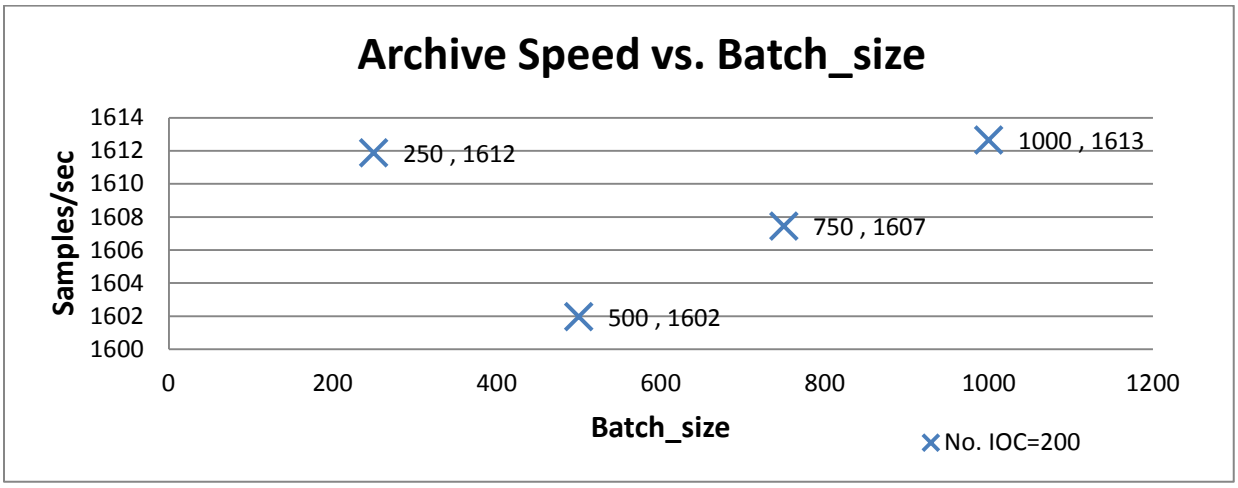
It is worth of noting that the archive speed is still increasing despite a slower growth rate, implying that the maximum archive speed may be optimized with other preference settings. Hence, the impact of `Write_period`, `Buffer_reserve` and `Batch_size=500` on the engine's performance are studied with the 200 IOCs. Graph 2, 3 and 4 shows the test result of respective parameters.



Graph 2. Archive Speed vs. Write_period

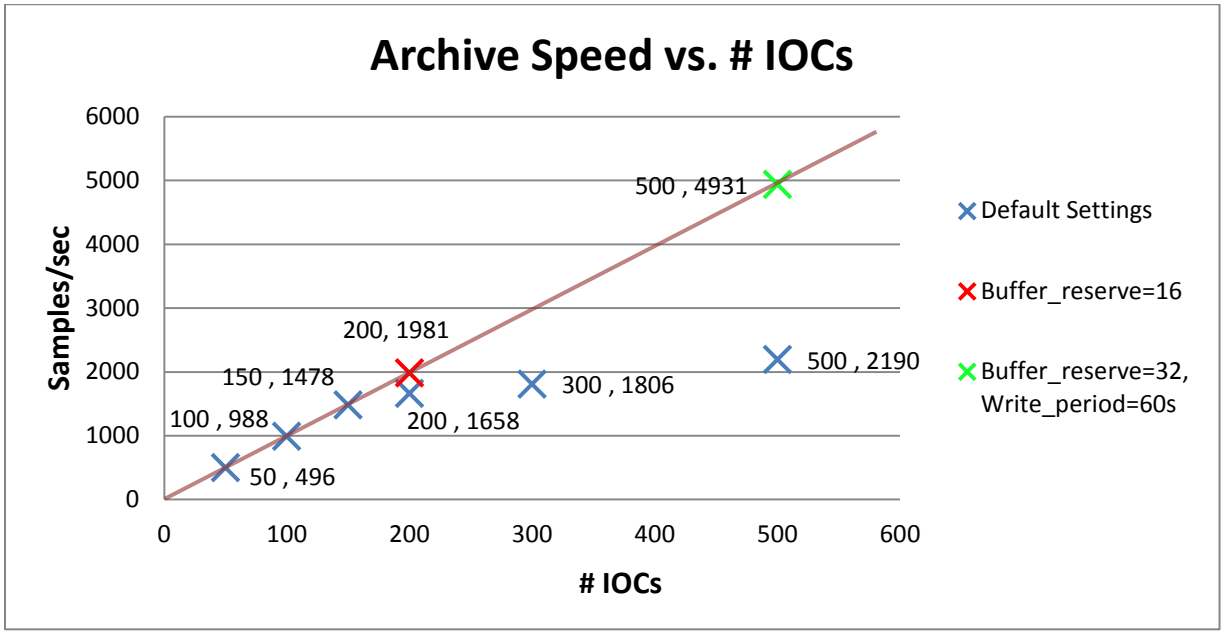


Graph 3. Archive Speed vs. Buffer_reserve



Graph 4. Archive Speed vs. Batch_size

The results show that both larger Write_period and larger Buffer_reserve have monotonic positive influences on the archive speed, but the influence of the Batch_size is unclear. It can be inferred that with larger Write_period and larger Buffer_reserve the archive speed capacity of the archive engine can be improved. Thus, the archive speed for 200 and 500 IOCs are tested again with larger Write_period and/or larger Buffer_reserve. The results are shown in Graph 5.



Graph 5. Archive Speed vs. # IOCs

The results are very exciting. The new data points line up with the three data points of 50, 100 and 150 IOCs and form a straight line. The archive speed capacity shoots up to as high as 5000 samples per second. Therefore, the archive speed capacity seems to be able to be improved at will.

However, there is a critical constraint in the experiment design: the sampling process only lasts 5 minutes. In operation, such archive engines need to run for days without a pause. Hence, we test the engine's capability to work for much longer periods. The results are summarized in Table 1.

No. IOC	Sample Time	Buffer_reserve	Sample Speed	Average Sample Size
100	1.02d=88128s	4	989/sec	74 Bytes
150	2.86h=10296s	8	1479/sec	76 Bytes
150	2.5h=9000s	16	1466/sec	75 Bytes
150	5.7h=20520s	4	1485/sec	76 Bytes
150	18.55h=66780s	4	1160/sec	76 Bytes
150	1.82d=157248s	16	1390/sec	76 Bytes
200	1.29h=4644s	8	1512/sec	76 Bytes
200	1.72h=6192s	16	1536/sec	75 Bytes
200	11.97h=43092s	4	1484/sec	76 Bytes

Table 1. Long Run Tests

It can be seen that when the working duration prolongs, the archive speed is affected. For 200 IOCs the archive speed slows down to about 1500 samples per second regardless of the buffer_reserve. For 150 IOCs, the speed decrease is also noticeable, where the speed slows down to 1390 samples per second in the 1.82 days run. For 100 IOCs, the engine is so far reliable for a full day run.

Consequently, although in the short term the archive speed of the engine can be very high, in the long term the engine tends to be sluggish. The estimation of the reliable long run archive speed is about 1000 samples per second, but this estimation need to be proved with more tests of longer working time.

One factor that limits the long term speed of the Archive Engine may be the size of memory of the Java Virtual Machine, which is currently 247.5 MB. The archive speed of the engine is dominated by the dynamics between the write speed and the accumulation rate of buffered sample. Only if the write speed is greater or equal to the accumulation rate can the archive engine sustain its speed in the long run. If a larger memory allows the engine to write faster, the archive speed will be improved.

The size of each sample remains relative constant at about 76 bytes.

Summary

The paper describes how the Archive Engine is built from sources and the related configurations for testing. The research only focuses on the MySQL database. The Archive Engine is tested with simulated PVs and is estimated to have a reliable long term archive speed of 1000 samples per second, although in the short term the speed can reach 5000 samples per second. For further researches, other factors such as the network traffic, the MySQL storage engine, the Java Virtual Machine memory and so on should also be tested extensively to find the most efficient combination of parameters. In addition, many other CSS tools that interface to the Archive Engine, for example the `AlarmServer`, are useful in real operations and should be built and tested.

This documentation serves as a setup and user's manual for the Archive Engine only. For more information regarding advanced technical details of the engine and other CSS products, please consult the CSS Guide and other related online documentation.

Appendix I. Related Web Links

1. EPICS homepage

http://www.aps.anl.gov/epics/index.php

2. SNS CSS web page,

http://ics-web.sns.ornl.gov/css/products.html

3. Source Forge repository

http://cd-studio.hg.sourceforge.net/hgweb/cs-studio

4. Eclipse Download page

http://www.eclipse.org/downloads/

5. Control System Studio Guide

http://cs-studio.sourceforge.net/docbook/

6. MySQL Download page

http://dev.mysql.com/downloads/mysql/

7. MySQL online documentation

http://dev.mysql.com/doc/index.html

References

1. K. Kasemir and Gabriele Carcassi. *Control System Studio Guide.* http://cs-studio.sourceforge.net/docbook/
2. K. Kasemir. *RDB Channel Archiver*. http://ics-web.sns.ornl.gov/css/docs/RDBChannelArchiver.doc
3. Personal emails with K. Kasemir.