

Advanced graphical presentation library

***Sgraph*** is an instant LabVIEW based data historian and presentation utility.

The objective of Sgraph is to provide a fast way to add cool graphic presentation tools for storing and analyzing large amounts of acquired data. Many tools can be tailored to provide user friendly and convenient visualizations, but many disadvantages of large datasets tend to appear frequently.

Sgraph is an attempt to provide a professional state of the art leap forward.

Fast  $\Rightarrow$  Intuitive  $\Rightarrow$  Scalable

André Koelewijn, 2020

Beta version V 0.4



[Sgraph reference document](#)

<b>1.</b>	<b>Sgraph fundamentals.....</b>	<b>3</b>
1.1	Data logging objectives narrative.....	3
1.2	Signal sampling Qtype definitions.....	5
1.3	Data historian requirements.....	8
1.4	Computer processing behavior narrative .....	9
1.5	Sgraph software architecture.....	10
1.5.1	Software abstraction levels.....	10
<b>2.</b>	<b>Sgraph library structure overview .....</b>	<b>11</b>
2.1	DataBase access Abstraction Layer (DBAL).....	12
2.2	Historian Write Interface (HISW) .....	13
2.2.1	Benchmark data source .....	13
2.2.2	Writing data to the historian.....	13
2.3	Fast Access Memory Manager (FAMM).....	15
2.3.1	Anti-Aliasing Layered datasets .....	15
2.3.2	The dataset object .....	15
2.3.3	TAG data element .....	16
2.3.4	The TAG Status definition .....	17
2.3.5	The TAGs memory element.....	18
2.3.6	The PAQ resolution layers.....	19
2.3.7	The PAQ size and display resolution .....	20
2.3.8	The PAQ resolution setting .....	21
2.3.8	The PAQ data manipulation .....	21
2.3.9	TAG properties .....	22
2.4	Averaging post processor (AVPP).....	23
2.4.1	Averaging span.....	23
2.5	View Content Solver (Vsolver) .....	24
2.6	Sgraph Server (Gserver) .....	24
2.7	Sgraph Client (Gview) .....	24
<b>8.</b>	<b>Abbreviations .....</b>	<b>27</b>



## 1. Sgraph fundamentals

The intention of Sgraph is to provide added value for storing repetitive acquired signals in a convenient and fast way. A long time is spent to determine if the historian should support waveform type, repetitive clocked, or event based data sources. The Sgraph objective is to support them all.

Author/Developer: Andre Koelewijn, 2020

### 1.1 Data logging objectives narrative

If someone starts to explain about data logging from one's own perspective, then the story tends to emphasize at the experiences and needs of the story teller. Every technical application field will have specific requirements and depending on products already available, the 'nice to haves' will be different for every situation. There are however common interests that can be standardized.

**Primary** the type of application will depend on the application field. Is it an DCS<sup>1</sup> system for chemical processing? Is it a test system to perform quality control for industrial production? Is it part of a MES<sup>2</sup> system for large scale industrial SCADA<sup>3</sup>? Is it a stand-alone DAQ<sup>4</sup> or ATE<sup>5</sup> system?

**Secondary** the need for logging can depend per application;

Is the test data needed to ensure proof for Airworthiness? Spacecraft safety? Medical grade equipment certification? Proof traceability? Ensure data integrity during calibration? Help engineers to troubleshoot? Provide insight for researchers? Perform statistical process control? Provide Business Intelligence? Aid software testing?

The above number of applications are far too wide to cover them all. Therefore, the Sgraph initiative is **not** meant to cover the structural 'application specific' needs. Those are part of the primary design specification. **Sgraph is meant to cover the need of the software developer** to have fast and immediate insight in acquired data. If such tools appear convenient for users: great! But primary it is intended for fast and immediate troubleshooting. In general: Have EVERYTHING available at ANY time to see WHATEVER one wants to see at probably another moment in time than when the required data was defined. Meaning: it is very common that only after the acquisition runs for a while, that a bug appears. Here it is very common that data and events in the past could reveal the cause of 'something' that happened. Most certainly this will be an 'aha' moment 'didn't think of that' where the identification of 'that' is unclear during preliminary design. So... Store everything indefinitely. Make 'close to perfect' tools to see everything always down to any moment in time. This can prevent many 'add logging and let it run again' iterations to find a cause of a problem.

My experience showed me that such is possible at mild costs. Modern drives have more than adequate space left to save data of all channels continuously for ever (i.e. have a 20 year retention or so...)

---

<sup>1</sup> DCS = Distributed Control System, by i.e. Siemens, Honeywell, Yokogawa, Emerson.

<sup>2</sup> MES = Manufacturing Execution System, large plant automation systems, very often based on batch control.

<sup>3</sup> SCADA = Supervisory Control and Data Acquisition,  
as smaller scope than MES, it can be regarded as a subsystem or predecessor of MES.

<sup>4</sup> DAQ = Data Acquisition System, often as subsystem of DCS, SCADA, MES or ATE.

<sup>5</sup> ATE = Automatic Test Equipment as part of Test & Measurement systems or product test & certification tooling.



Selecting ‘what to see’ afterward on such systems can be a very satisfying experience. On the sideline it’s possible to regard such data as a nice ‘data lake’ source and store them in a versatile structured manner. Having both an interface, a structured storing method and a data description can be a convenient data source for SPC or BI tools developers. It can be regarded as just a more advanced way of troubleshooting called optimization.

### ***Calibration Status***

The calibration due dates and calibration settings are not part of the functionality of Sgraph but considered part of the acquisition system. Although it is provided for the data element to store calibration status to provide the potential to review the status at hindsight. After all; viewing them could appear to be very interesting.

### ***Alarm threshold settings***

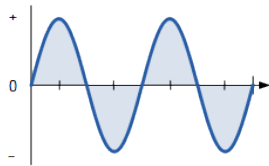
The threshold at which an alarm will occur is not considered a functionality that belongs to Sgraph. This is considered a function of the processing system (like DCS or MES). It is provided that the data sent can contain the Alarm status so it can be stored together with the data. After all; viewing this could also appear to be very interesting.

## 1.2 Signal sampling Qtype definitions

To start explaining the technology used it is important to first get the vocabulary right using a defined signal definition. There are three different **Qtypes** of acquisition defined for Sgraph:

- Waveforms. (WAQ)
- Periodic acquisition. (PAQ)
- Event based signals. (EAQ)

Every one of those signals have different properties and ask for a different approach on database design to be most efficient. Please be aware that these descriptions are not to be interpreted as general descriptions, but to describe three types of acquisition as supported by Sgraph and being the three **Sgraph methods** to support a vast number of different applications.



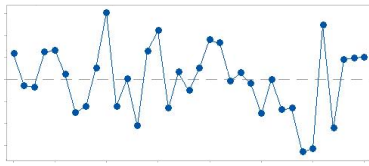
### Waveforms (WAQ)

A waveform can consist of one or more simultaneously sampled signal channels where the first sample is defined by time  $T_0$  and every subsequent sample added to form a 2D array where  $dT$  defines the time between subsequent samples. Apart from  $T_0$  The separate samples to not have a timestamp. The time for each individual sample is

determined by the sample count in respect of  $T_0$  when one takes into account the  $dT$ .

This kind of sampling is used where usually a limited amount of samples is used for a defined time period. This can be in example the sampling of a movement during a few seconds, an oscilloscope or sampler acquisition or serial data waveform shape fetch. where the sample rate can be as high as the hardware and FIFO<sup>6</sup> buffer can deliver. Although continuous acquisition is usually an option for a waveform signal Qtype, this is not included here since this kind of sampling is usually meant to be used for very high sample rates (i.e. 1Gs/sec) where it is expected that a database cannot keep up to streaming and storing all data continuously. Another reason to limit this type of fast acquisition not to be continuous is that one single  $T_0$  value does not guarantee absolute time information if a long time has passed. It does not provide time synchronization other than the waveform start event.

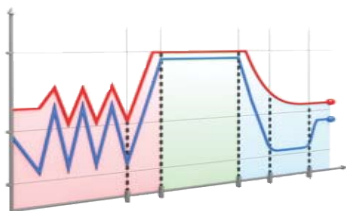
The difference between a waveform and periodic continuous acquisition is the distinction that a waveform makes use of a FIFO to acquire fast sampled data for a limited amount of time.



### Periodic Acquisition (PAQ)

For periodic (continuous) acquisition there is also an array of channels used just like a waveform. To clear out the vocabulary used here; with periodic acquisition we mean periodic continuous acquisition, the word ‘continuous’ is sometimes left out. With periodic acquisition every sample that contains one or

more channels contains its own unique timestamp. This property is such that the subsequent channels do not need to have a strict deterministic sample time. An ‘approximate’ value for the sample rate as generated by a non-deterministic operating system can be adequate to perform the task. Since every sample of one or more channels has its own timestamp, the time of each signal in time is known. For this kind of measurements however it is necessary that all channels are sampled simultaneously. If all measurement channels have its own unique sample rate, then this type of sampling is not efficient. If simultaneously sampled, then the sample rate of all channels is defined by the speed of the slowest channel. Such kind of sampling can be used for medium speed data acquisition systems and DCS<sup>7</sup> systems. Sampling rate is normally expected to be between 10 ms and 1 minute. When sampling speeds are above 10 ms then usually realtime acquisition hardware is used (i.e. a RT-cRIO<sup>8</sup> system).



### Event based sampling (EAQ)

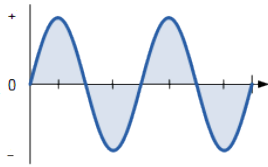
When the several signal sources are not synchronized in such a way that every signal is able to arrive at any random moment in time then event driven sampling should be used. For event signals every single sample has its own unique timestamp. The advantage of the periodic sampled acquisition is that every signal source can be processed independent. The disadvantage

can be that there is more overhead if one wants to retrieve large amounts of data since every sample has to be retrieved independently instead of a whole range of channels simultaneously. Therefore, the costs of event based sampling instead of periodic ‘array wise’ acquisition is that browsing through large datasets can take more access time from a database. Also this type of data acquisition is less efficient in terms of database footprint. It will result in a larger database. Another disadvantage from periodic acquisition is that there is not a defined ‘time since last sample’ available so therefore it is more complicated and expected to be unsupported to have anti-aliasing and min/max readings. For events however that are expected to occur only for a limited number in time, like configuration change events or configuration switch settings an event based signal can be very helpful and save much overhead. It would be a waste of resources to sample a switch setting 10 times per second if one knows that it will only change once per day.

<sup>7</sup> Distributed Control System, think of a vast number of PID controls in a large chemical plant control system.

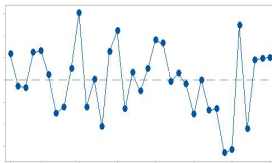
<sup>8</sup> National Instruments cRIO chassis with LabVIEW Realtime operating system.

So Just for completeness, let's list the main properties of each sampling Qtype:



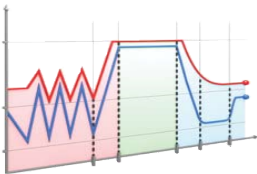
○ **WAQ**

- Very fast acquisition (around 10 ks/sec up to 10 Gs/sec)
- Finite acquisition time (i.e. 10 seconds)
- Timing defined by  $T_0$  and  $dT$  (sample speed)
- All channels sampled simultaneously
- Deterministic (usual very accurately timed)
- Typical application field: DAQ and ATE systems



○ **PAQ**

- Medium speed acquisition (around 6 s/min up to 10 s/sec)
- Continuous acquisition
- Usual fixed acquisition speed
- One timestamp per sample of multiple channels
- All channels sampled simultaneously
- Usual non-deterministic (not real time)
- Typical application field: DCS or MES systems



○ **EAQ**

- Slow or incidental acquisition
- One channel, one sample at a time
- Individual timestamp per sample
- Every channel sampled individually
- non-deterministic
- Typical application field: ATE and MES systems

### 1.3 Data historian requirements

The objectives of this Sgraph database historian evolved with growing insight when developing systems varying from small slow sampling systems, medium samplers for deterministic DCS systems with high channel count, PLC based MES systems with independent data sources, up to hi-speed analyzers with hardware based FIFO buffers.

A common requirement for all those systems is lots of data and the desire to check out this data fast and have a user friendly interface to do such. When such systems are built frequently using a similar architecture then standardizing the acquisition and data storage is a recurring requirement.

Having seen a lot of very different applications a shared set of requirements keeps appearing. Therefore, the Sgraph initiative is to develop one library of easy to implement VI's that enable all of the common requirements. With my gained experiences I now attempt to improve them into one better solution. Add the library, pick your acquisition method (as you have them) and connect them to the historian code. Then just use the standard visualizer to browse over the samples, zoom from years of data down to milliseconds in not more than a throw at the scroll wheel. Minor latency under the hood of a tool with a hopefully delightful user experience.

Main objectives of Sgraph:

- Sample indefinitely (> 20 years)
- Acceptable database size
- Easy to implement
- Support three Qtypes of acquisition (waveform, periodic, events)
- Separate database layer (SQL, TDMS, BIN, HDF5, Custom)
- Support local retrieval without dependency of commercial purchased drivers
- Fast retrieval (< 1 sec)
- Automatic anti-aliasing and min-max envelope view
- Support multiple simultaneous independent operating viewers
- Viewer: fast zoom & scroll
- Viewer: support areas of interest and channel combinations
- Viewer: fast normalizing, selecting, scaling
- Viewer: easy export (excel, csv, graphic)
- Viewer: intuitive JIT help (no manual required)
- Viewer: Support multiple channel adapting scales
- Viewer: Support up to 8K EHD screen resolutions
- Viewer: auto-adapt viewer resolution for optimal detail
- Option: open interface to integrate custom functions
- Universal integrator expertise (dummy setup or specialized dataflow expert setup)





## 1.4 Computer processing behavior narrative

The computing power and processing focus of Sgraph follow a distinct set of principles to ensure an optimal user experience. To begin with the Sgraph viewer should be very responsive and should not show noticeable latency. Since some serious data processing is involved this could be a hard job to perform. At least it should be clear that any HMI<sup>9</sup> interaction should be immediate to give the user an experience of direct responsiveness. Since the AVPP<sup>10</sup> must perform calculations before the average and min/max data is stored, this can cause latency if one is watching this data live or short after acquisition. But still this processing has lower priority than keeping track of the acquisition buffers of the HISW<sup>11</sup> module to prevent overflow of acquisition buffers.

Overall direct behind the main priority of the viewer HMI event handlers, the next subsequent priority is given to the data server in the form of the Gserver<sup>12</sup> where it's submodule Vsolver<sup>13</sup> will determine what dataset is used, and deliver this data with flashing speed to the FAMM<sup>14</sup> which is optimized to deliver the result of the Vsolver query as soon as possible using every computing power possible. Since modern operating systems have an abundant processing power, this won't be any problem. The FAMM is nearly an interface to the PC's RAM<sup>15</sup> memory which will act as fast cache memory so the system doesn't have to wait for disk or network access.

The multithreading priorities can therefore be classified as follows:

1. GUI interactions → User perception of 'immediate' response (i.e. zoom & scroll)
2. HISW → Prevent buffer overflow to support deterministic DAQ behavior
3. Vsolver / FAMM server action → User perception of 'very fast' data fetch & view
4. AVPP → Make sure this is processed before storing on disk
5. DBAL → Store data when all other stuff is done

The result should be that the user can have any dataset combination fetched, pan & zoomed from milliseconds up to several months at flashing speed with hardly noticeable latency. The fast RAM access should do the immediate actions. The only time when the Vsolver has to retrieve data directly from disk is if a user wants to view short timespans further back in time, i.e. 3.5 seconds of data that occurred 3 months ago at a certain moment in time. But even that action shouldn't take more than an average database access query and be well below one second.

Think of being zoomed in to your own house on Google Earth, and then zoom out with your scroll wheel until you see the entire globe. That idea, but then a little faster.

---

<sup>9</sup> HMI = Human Machine Interface, another name for Graphical User Interface or GUI

<sup>10</sup> AVPP = Averaging Post Processor, the module that takes care of min/max and averaging.

<sup>11</sup> HISW = Historian Write interface, the high level module that is used to write data.

<sup>12</sup> Gserver = Common set of code that delivers data to the Sgraph viewer

<sup>13</sup> Vsolver = module that determines which data set is needed by the Sgraph

<sup>14</sup> FAMM = Fast Access Memory Manager, the RAM portion of the data historian

<sup>15</sup> RAM = Random Access Memory = the computer's fast onboard working memory.









## 1.5 Sgraph software architecture

The Sgraph software makes use of native LabVIEW object oriented software. In other words, classed that use data by value. The Sgraph project did not use any GOOP tools. There is however one location where referenced data pointers are used; the FAMM class. The FAMM class makes use of one memory class object per resolution layer. So if the system has 8 resolution layers; there are 8 memory class instances. Every memory class instance will access the memory with a fixed set of DVR objects that are utilized 'by reference' to prevent duplicate data copies and prevent race conditions. Since the DVR 'in place element structure' takes care that such an object cannot be accessed twice, a deadlock condition (software will hang) or runtime error will indicate invalid design. Since the access to the memory objects are deep down hidden in the software architecture, once proper programmed it should work fine.

### 1.5.1 Software abstraction levels

To show programmers when programming how deep they dig, there are several icons that are pasted into the diagram of the LabVIEW vi's to make the programmer aware at which abstraction level they navigate.

	Abstraction Level 0 First Sub-vi to call with code that can be handled by every LabVIEW programmer. Common code blocks to be used at will.
	Abstraction Level 1 Now you went one subVI deeper into the system, for experienced programmers no problem, for beginners: be careful here.
	Abstraction Level 2 Now you're digging deeper, don't fuzz around here if you're not an experienced LabVIEW programmer. Here some fundamental functionality is present.
	Abstraction Level 3 You're entering the basic building blocks of Sgraph, don't change anything here if you don't understand the design pattern architecture. Expert stuff only.
	Abstraction Level 4 The foundation blocks of Sgraph. Only expert stuff here. Lots of code and system performance depends on this. No playing around here.
	Abstraction Level 5+ Should not exist. And if it does then I guess that more complexity is needed. Arrow shows 'you're dangerously deep' and 'might not be necessary'. Can also be regarded as the overpressure indicator of a submarine.

## 2. Sgraph library structure overview

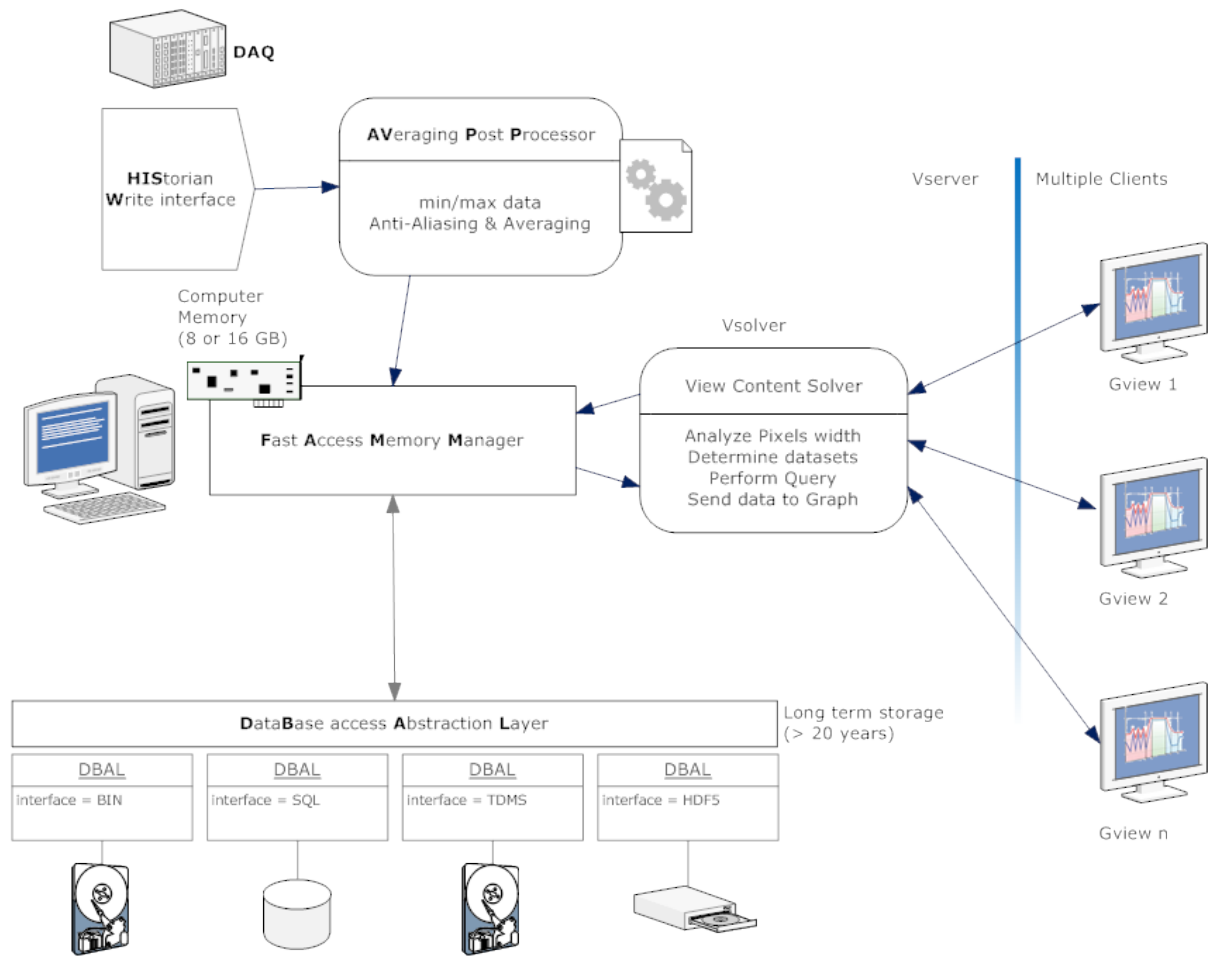
The Sgraph library must be easy to add, and therefore is available as a national Instruments project or native library (lvlib). The library is intended to become object oriented since the expectation is that the project is to become far too complicated to depend on a user to maintain the data structure integrity without some mechanism to prevent accidental use of code outside the scope and ensure reliability. Since it might become a multi developer open source project this also directs towards object orientation.

The several subsections can rely on a vast number of previous built data loggers leading to an optimal way of componentizing the various subsections for maximum maintainability and scalability.

1. Database access abstraction layer (DBAL)
2. Historian Write Interface (HISW)
3. Fast Access Memory Manager (FAMM)
4. Averaging post processor (AVPP)
5. View Content Solver (Vsolver)
6. Sgraph Server (Gserver)
7. Sgraph Client (Gview)

This list is by far not enough to describe all details, and some of the above mentioned sections are much larger than others. It does however contain about the number of building blocks that were found very useful in previous projects to focus on separately, and helping to gain insight in a functional description of the whole.





## 2.1 DataBase access Abstraction Layer (DBAL)

The database abstraction layer plays a role in maintaining a standard interface between data storage on disk and the Sgraph application. Since type of database can vary per designer preference or other circumstances, this abstraction layer or driver makes sure that data storage matches your needs. There might be more needs or other frontends that you would like to use the same data as saved by the Sgraph historian, and this DBAL makes it possible. The first applications I used were based on BIN type data storage but most certainly an interface to any database type can be implemented.

The reason that I would like to test multiple different types of storage is to gain experience in the benchmarks between several types. An abundant amount of information can be found online in comparing different storage and database types with all having their individual pros and cons. My first focus will probably be SQL versus BIN to compare speed performance and latency.

By expectations are as follows:

- BIN fast and no need for any third party driver (like MySQL or SQLite).
- SQL fast and universal access of the data by other front ends.
- TDMS advised and versatile for LabVIEW users without need for drivers.

## 2.2 Historian Write Interface (HISW)

The universal input interface for historian Sgraph historian data is the Historian Write Interface (HISW). This high level interface is the primary input to periodically save data into the historian. It is the connection between the data acquisition system (or MES or DCS system) and the Sgraph data historian.

### 2.2.1 Benchmark data source

In order to develop and test the Sgraph library a standardized dataset is needed to act as a benchmark. With such a standardized data source it is possible to test various different configurations (i.e. different DBAL's) or the performance speed and processor load of various operating systems.

The benchmark data source makes use of the following dataset:

- 1000 tags of PAQ with an acquisition speed of 10 samples/second
- 32 tags of WAQ with length of 10k samples of duration and 50ks/sec speed and one waveform at a time, updated 1 waveform per second.
- 1000 tags of EAQ each randomly updated between 1 and 60 seconds per sample.

This continuously generated benchmark data should be allowed to be run and saved in both the AVPP, FAMM and DBAL with acceptable processor load, while allowing for uninterrupted parallel processing of at least four instances of a Gview server by the Vsolver. This total Sgraph system operation must be able to run with an acceptable processor load and computer memory footprint so other main applications (like MES, DCS or other DAQ or SCADA system) can run without noticeable latency. Also the general OS functions should be able to run on such systems without problem.

### 2.2.2 Writing data to the historian

The HISW (Historian Write Interface) has to provide for three different Qtypes of data. For each of those data Qtypes there is a different interface, so in fact there are three HISW's. All three of them however are present in three public methods of the HISW class.

A Acquisition system user (client) calls a public method from the HISW class in order to write data to the historian. The HISW takes care that the received data is sent to the AVPP for processing by the FAMM. From the FAMM the decision is taken whether or not write to the database. For the HISW however; only the AVPP has to be addressed.

The user does not have to mind about the configuration of the database and it's TAG's. When writing data: the HISW will check if all TAG's exist. If not: then it will extend the existing database with the TAG's it sends. This 'auto-grow' mechanism takes more time than the design latency of the system, but this is only during configuration changes, after the first configuration change the system will again boost to full speed.

The HISW is a child class of the AVPP (Averaging Post Processor) where a call to store one array (one sample of all TAGs to send) is nearly a call with a formatted content to the AVPP. Where to store the data and averaging an array of data from each resolution layer and writing an averaged and min/max determined sample into the next lower resolution layer is performed by the AVPP. Opening the single AVPP call opens up the writing side of the complexity of the system. The only extra that the HISW performs is adding TAGs to the configuration with default setting if the HISW wants to write non-existing TAGs.





## 2.3 Fast Access Memory Manager (FAMM)

The Fast Access Memory Manager (FAMM) is the RAM memory manager of Sgraph. If the Vsolver performs a query to the FAMM then it will check if the content is present in the FAMM. The FAMM will then deliver a dataset to the Vsolver at its highest speed. Since the FAMM is also used to continuously store data as received from the AVPP it is needed that this piece of code performs at with high efficiency and therefore have a well-designed software architecture to save both memory footprint prevent excessive CPU load.

### 2.3.1 Anti-Aliasing Layered datasets

One of the key building blocks of Sgraph is the used of layered datasets. To create fast responsiveness of the Gview displays if large dataset are requested; the Sgraph database makes use of tailored datasets for every timespan range. The fastest recorded data is transformed by the AVPP (averaging postprocessor) into lower resolution datasets. These lower resolution datasets make use of three data points per sample:

- Averaged value
- Maximum value
- Minimum value

With the above three values an image of the averaged data can be shown, where the max and min values (of the higher resolution signal) can be used to visualize a background envelope solid with a ‘half transparent’ lighter taint to indicate an effect as shown on analogue oscilloscopes. In this way the user is aware of ‘higher frequency’ data while showing an anti-aliased averaged center data line. This is the effect that some equipment manufacturers identify as ‘digital phosphor’. The anti-aliasing effect is realized by using a good compression ratio for every lower resolution dataset. Such lower resolution datasets are best to be kept to maximum 1:10 ratio.

For WAQ and PAQ datasets the resolution steps can be designed due to the nature of the signal as having a defined sample speed. Due to the unexpected nature of event based EAQ signals the resolution cannot be exactly identified for EAQ. Here only the highest resolution can be defined, where the higher resolution datasets will stay empty if not used, and contain similar data in the lower resolution datasets.

### 2.3.2 The dataset object

The various datasets as described in par 2.3.1 should have and efficient footprint and prevent multiple copies to support both small memory footprint and prevent race conditions. Therefore the several RAM memory elements of the FAMM are dynamically created and use ‘by reference’ read-write operations. By using a DVR and making functional FG variables for access control.

### 2.3.3 TAG data element

The basic data element for holding a sample (any Qtype) in memory is based upon the DAQ variable name. This variable name is called a TAG. This TAG name can have any logical name as used in the control system. These can differ per naming convention and are irrelevant for the functionality of the software.


#### ***TAG name dependent functionality***

There are existing systems where a distinction in functionality is formatted in the TAG name. In example; the difference between a Setpoint (SP), a Process value (PV) or Output value (OP). To support such a convention without losing the universal character of Sgraph, functionality bits are provided in the TagFunc binary word. By filtering of specific functions on those bits (which might be correspond to a certain TAG naming format) such naming format functionality is provided but not forced. These TagFunc bits reside within the TagProp array. The designation of TagFunc is defined in the FuncDesc constant.

#### ***The TAG Sample format***

Sample

Tag	Time	Source	status	data	max	min
	0		00000000	0.00	0.00	0.00



1. **String** TAG name, which is essentially the name of the data element. The TAG definition contains additional properties this tag regarding its type, description, range and several additional functional properties. The TAG properties are never changed but loaded at startup.
2. **DBL** The time of the last modification on this tag by the source as identified by (3)
3. **String** The last software method that performed actions on this TAG i.e. ramping, Control algorithms, Human data entry (GUI), Alarm interlock etc. This string can act as very important dataflow mechanism if TAG's can be controlled by several methods which is very common. By using a string, the tracking of a TAG throughout complex software is very convenient. If no extra operation is performed then this is usually DAQ (acquisition system).
4. **U32** The binary status bit. It holds Alarm status, sensor status, Interlock status, etc.
5. **DBL** The data of this sample, will be DBL for any sample data type.
6. **DBL** The max value from the next upper layer of higher resolution samples.
7. **DBL** The min value from the next upper layer of higher resolution samples.

The above TAG format is the universal way to transport a single element of data throughout the system. Where multiple TAGs are communicated then it will be an array of TAGs. An exception is for the FAMM memory element. Where the FAMM makes use of the above format to communicate with any external object, the internal representation of a FAMM memory object is slightly different due to memory footprint and processing speed efficiency.

The above TAG format is defined in a typedef in: `Sgraph\source\Common\controls\typedef_com_OneTag.ctl`





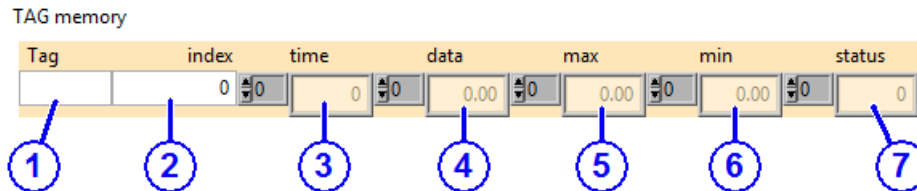
### 2.3.4 The TAG Status definition

The Status [U32] binary word needs some special attention since it's the placeholder of several types of extra circumstantial information that can occur on any moment during acquisition or subsequent processing steps. It is possible than an acquired sample is unreliable, close to its upper range, above an alarm value, out of calibration, or locked out by an Alarm or human intervention action All such data is live sent with the TAG in the Status but throughout it's travel across all processing steps. The table below defines the meaning of the several bits of the status word.

Bit	Function	Application(s)
0	Enabled	0= In case sensors can be powered off, otherwise =1
1	Valid	1= Value can be used, all OK
2	Error	1= Indication that sensor is defective
3	Over range	1= Input signal was higher than positive upper range
4	Under range	1= Input signal was lower than negative upper range
5	Max/min data present	1= max/min data present, 0= ignore max/min data fields
6		Spare (sensor quality bits 0..7)
7		Spare (sensor quality bits 0..7)
8	H Alarm	1= An alarm threshold is reached or passed
9	HH Alarm	1= An alarm threshold is reached or passed
10	HHH Alarm	1= An alarm threshold is reached or passed
11	L Alarm	1= An alarm threshold is reached or passed
12	LL Alarm	1= An alarm threshold is reached or passed
13	LLL Alarm	1= An alarm threshold is reached or passed
14	Accumulated Alarm	1= One of the above Alarm conditions is True
15		Spare (alarm annunciations bits 8..15)
16	Interlocked	1= Safety interlock control is active (i.e. by Alarm circuitry)
17	Manual	1= Manual control active (if applicable for control loops)
18	Alarm acknowledged	1= If bit 14 alarm is acknowledged by an operator
19	Alarm override	1= if Alarm functions of bits 8..13 are disabled
20		Spare (Interlock bits 16..23)
21		Spare (Interlock bits 16..23)
22		Spare (Interlock bits 16..23)
23		Spare (Interlock bits 16..23)
24	Ramping Active	1 = ramping towards end value
25	Signal stabilized	1 = sensor or control signal is steady
26	Calibration expired	1 = Channel calibration due date expired or not valid
27		Spare (Miscellaneous bits 24..31)
28		Spare (Miscellaneous bits 24..31)
29		Spare (Miscellaneous bits 24..31)
30		Spare (Miscellaneous bits 24..31)
31		Spare (Miscellaneous bits 24..31)

### 2.3.5 The TAGs memory element

Although the TAG definition of paragraph 2.3.3 shows a complete TAG dataset, to read and write these Tags from memory another data format is needed. The memory element that is used for fast reading and writing does not need to define the TAG name or Source info for every sample. But it does need a fast array search function where LabVIEW proofs to be able to search fast if only 1D arrays are used. Therefore the TAG memory structure for the FAMM memory is as follows:



1. **String** The TAG name as defined in TagProp global constant
2. **I32** The current index position of the circular buffer that is created with this object. this is in fact a pointer of the buffer location of (3) up to (7)
3. **[DBL]** Array of timestamps of (4), (5), (6) and (7)
4. **[DBL]** Array of datapoints, this is the primary data
5. **[DBL]** Array of maximum peak values of upper higher resolution arrays, empty if not used
6. **[DBL]** Array of minimum peak values of upper higher resolution arrays, empty if not used
7. **[U32]** Array of status words that belong to (4) per definition of par. 2.3.4

The format as shown above is not used 'by value' but referenced using a LabVIEW DVR. In the constructor of the FAMM a number of instances of this element is created. It is created for as much TAGs as exist in the system and the references to these objects are stored as FAMM attributes.

The above TAG memory format is defined in a typedef in:

Sgraph\source\Common\controls\typedef\_com\_TagMem.ctl

### 2.3.6 The PAQ resolution layers

In the constructor of the FAMM the memory objects per different Qtypes are instantiated. The number of anti-aliasing layers for every Qtype depend on several historian constraints. One of the constraints is the expected sampling rate for the PAQ (periodic acquisition) signal type. Where WAQ samples are defined per waveform and EAQ events do not have a fixed rate at all, the PAQ Qtype does have a continuous update rate and therefore needs a fixed set of anti-aliasing layers. If the historian is used for the first time and not used before: then the PAQ channel definitions in combination with the PAQ channel list will determine the required number of layers. Since for the several layers the 'timespan' determines its behavior, and a user's view is also referred to this 'time' resolution; the layer resolution is not determined using a ratio 'per layer'. Instead our Julian calendar system for time resolution makes it convenient and intuitive if the time slices are synchronized with our clock definitions that are human recognizable. That means that the resolution idents are based on 'milliseconds', 'seconds', 'minutes', 'hours', 'days', 'weeks' and 'months'.

Therefore the PAQ sampling system make use of the following layered averaging system, where the upper layers (L0, L1 etc) are removed if not used.

PAQ Layers for several time resolution systems

Time resolution of the PAQ data					
10 msec	100 msec	1.0 second	10 seconds		
Layer # as defined per system				Sampling system per layer	Table name
L0				10 millisecond per sample	PAQ_10mil
L1	L0			100 millisecond per sample	PAQ_100mil
L2	L1	L0		1.0 second per sample	PAQ_1sec
L3	L2	L1	L0	10 seconds per sample	PAQ_10sec
L4	L3	L2	L1	One sample per minute	PAQ_1min
L5	L4	L3	L2	One sample per 10 minutes	PAQ_10min
L6	L4	L4	L3	One sample per 2 hours	PAQ_2hr
L7	L6	L5	L4	One sample per day	PAQ_1day
L8	L7	L6	L5	One sample per week	PAQ_1wk

9	8	7	6	← number of Layers
---	---	---	---	--------------------

The L# numerical value for each resolution layer is in fact the index of the array of one of the objects that is instantiated by the FAMM constructor.

The **Sampling system per layer** is based on an empirical ratio of maximum 1:10 that will minimize memory footprint for Gview queries and provide an excellent anti-aliasing optimized viewport on the data. One of the objectives after all is to have flashing zoom speed over a very large dataset.

**Note:** the time resolution is set during historian setup. It cannot be changed without installing a new historian database. The time resolution is a primary design constraint and should not change during operation.

### 2.3.7 The PAQ size and display resolution

One of the main objectives is the ability for Gviewer to support high resolution displays up to 8K and provide perfect matched images on it. For the FAMM to support that and deliver instant data from memory; the size of the memory should be capable of such. At this point only the horizontal display resolution matters. The table below shows the several display resolutions currently defined.

Display resolution	Horizontal resolution	Necessary FAMM memory per layer
Full HD 1920 x 1080	1920 pixels	8k samples
4K 4096 x 2160	4096 pixels	16k samples
8K 8192 x 4320	8192 pixels	32k samples

The right column shows more samples than pixels to enable the user to scroll back in time a few times more than the current selected resolution without extra disk access time. Previous applications used 16k samples as an adequate buffer size with acceptable memory load and performance. Although the Sgraph design makes use of larger datasets due to added functionality, so did Moore's law add memory to the average workstation. Therefore the sample depth per layer is set to 32k samples. For a 9-layer FAMM system that would mean: 32k x 68byte = 2Mbyte per highest resolution layers.

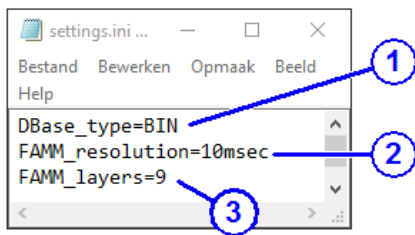
Available timespan and memory footprint per layer:

Layer	Samples	Memory	Timespan max	Timespan HD
PAQ 10mil	32k	2 Mbyte	5.3 minutes	19.2 seconds
PAQ 100mil	32k	2 Mbyte	53 minutes	3.2 minutes
PAQ 1sec	32k	2 Mbyte	8.9 hours	18 minutes
PAQ 10sec	32k	2 Mbyte	3.7 days	3 hours
PAQ 1min	32k	2 Mbyte	22 days	18 hours
PAQ 10min	32k	2 Mbyte	32 weeks	1 week
PAQ 2hr	32k	2 Mbyte	7.3 years	3 months
PAQ 1day	8k	0.5 Mbyte	22 years	3 years
PAQ 1wk	1k	low	20 years	21 years
Total	233k	15 Mbyte	(per TAG)	

For above examples one should realize that a full HD graphical presentation is already extremely high. Usually a presentation with a width of 300 pixels is adequate. The overkill according above table therefore is more than adequate to support abundant quality for many years to come. So even when a system would be able to sample for the above mentioned 20 years... probably the supported resolution would still be impressive.

### 2.3.8 The PAQ resolution setting

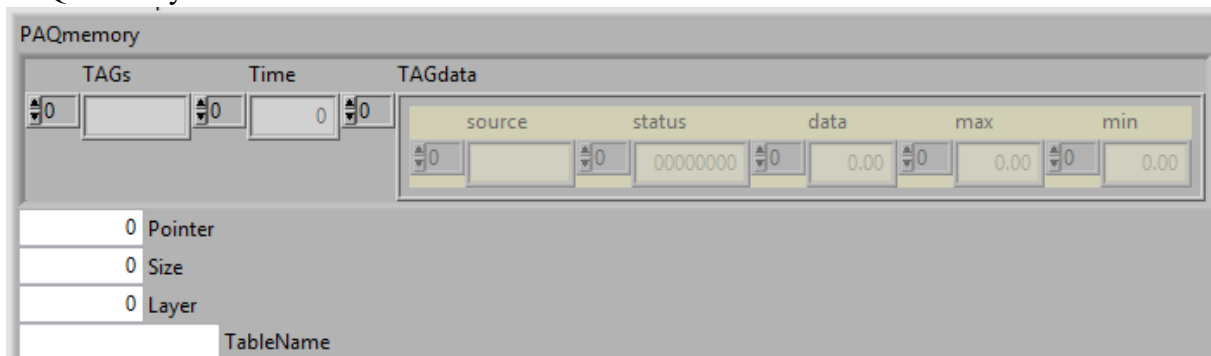
The settings and definitions of the FAMM and DBAL operation on the PAQ Qtype layers are defined in the settings.ini file located in: \Sgraph\database\almanac\settings.ini



1. Database type, BIN, SQL, MDB
2. Time resolution of PAQ data in FAMM layer
3. Number of layers (tables) of PAQ time resolution

### 2.3.8 The PAQ data manipulation

Reading, writing and initialization of the PAQ data is being performed by the FAMM class. The FAMM class contains several instances of the PAQ-memory class. One for every resolution layer. For an 9-layer PAQ Qtype memory there are 8 instances of PAQ-memory. If an acquisition system has the PAQ time resolution set to 100msec then there are 8 instances of PAQ-memory and so on.



The Pointer is to identify the circular data location from [Time] and [TAGdata]-elements.  
The Size is needed to determine the 'carry' position for the Pointer (back to zero).  
The Layer integer is the identifier of which L0 .. L8 instance this object refers to.  
The TableName is the name as defined in par. 2.3.7 and is also used for SQL or BIN files.

The PAQmemory shows that the TAGs are independent of TAGdata to empower fast search over the 1D TAGs array for read/write actions per TAG id.

One might observe that there is only one [Time] array. Please realize that the [Time] array does NOT equal the [TAGs] Array, but is equal in size of every array size of one of the elements that reside within one [TAGdata] element. In example; the 33<sup>rd</sup> element of TAGs contains the name of the content of the 33<sup>rd</sup> element of TAGdata. If someone would like to know the data of the 200<sup>th</sup> element of the buffer, then the 200<sup>th</sup> element of all elements of the 33<sup>rd</sup> element of [TAGdata] should be fetched. For the timestamp of every one of that 33<sup>rd</sup> element of [TAGdata], the corresponding 200<sup>th</sup> element of [Time] should also be retrieved.

### 2.3.9 TAG properties

Every TAG can have distinct properties that are initialized during startup of the system and retrieved from the [TagProp] array where each array element holds the properties of one tag. For a TAG to exist: is MUST have one TagProp element.

TAG properties:

Parameter	Type	Description
Description	String	Overall description of the TAG, can be used for online HELP
EngUnits	String	The Engineering units (°C, mV, kg, A, μT, etc)
RangeH	DBL	Upper range for 0 – 100% indication
RangeL	DBL	Lower range for 0 – 100% indication
Resolution	I32	Resolution of the D/A converter in bits, -1 means unknown
Obsolete	Bool	1 = not used anymore (skip this TAG), 0 = active
ChangeDate	Time	Date at which the data of this TagProp has been changed
ChangeSource	String	The software part or name of the person that changed this data
ChangeReason	String	Some explanation why this tagProp data has been changed
TagFunc	U32	Binary presentation of special TAG functions

#### ***TAG name dependent functionality***

There are existing systems where a distinction in functionality is formatted in the TAG name. In example; the difference between a Setpoint (SP), a Process value (PV) or Output value (OP). To support such a convention without losing the universal character of Sgraph, functionality bits are provided in the TagFunc binary word. By filtering of specific functions on those bits (which might be correspond to a certain TAG naming format) such naming format functionality is provided but not forced. These TagFunc bits reside within the TagProp array. The designation of TagFunc is defined in the FuncDesc constant.

## 2.4 Averaging post processor (AVPP)

The Averaging Post Processor (AVPP) has the task to perform data processing when a sample is received from the HISW before it is stored into the FAMM. The intention is to realize a few stacked layers of sampling information, each having its dedicated time resolution. The objective to do such is to allow a request for data to be answered very fast, where the Gviewer's data request will result in a dataset that is optimized for the time resolution set at that graphic presentation. To prevent data to be sampled and processed from a large dataset when needed, this processed data is already present in memory. That processing 'just in time' is performed 'sample by sample' during acquisition time by the AVPP.

### 2.4.1 Averaging span

The resolution layer time slices are defined according to the table in paragraph 2.3.6 and give an impression of the available resolution choices for the Vsolver. But what if one of those higher layer samples needs to be interpreted. How does one sample represent the Averaged and min/max value of a lower level? One is tempted to jump to a conclusion directly, but the real interpretation needs some more attention to let the value represent an intuitive correct value. So show how to interpret the higher level samples, and to let the programmer understand how the samples are saved into the FAMM and historian database, the table below shows the representation of the several samples.

Table resolution	Interpretation at time T		Concept
PAQ_10mil	T = as defined in HISW		One sample
PAQ_100mil	From T – 50 msec	to T + 50 msec	Centered span around T
PAQ_1sec	From T – 0.5 sec	to T + 0.5 sec	Centered span around T
PAQ_10sec	From T – 5 sec	to T + 5 sec	Centered span around T
PAQ_1min	From T – 30 sec	to T + 30 sec	Centered span around T
PAQ_10min	From T – 5 min	to T + 5 min	Centered span around T
PAQ_2hr	From T – 1 hr	to T + 1 hr	Centered span around T
PAQ_1day	From 01:00	to 23:00	The whole day span
PAQ_1wk	From Sunday	to Saturday	Week number

For the PAQ\_1day table the 'whole day' span makes use of the previously PAQ\_2hr samples and therefore has a slight deviation of one hour. This combines programming efficiency with a neglectable time shift of 0.04% where one should also take into account minor differences during daylight saving time. The timestamp for this table will use 12:00 in a 24 hrs clock system; so halfway the day.

For the PAQ\_1wk the week definition of Sundays.. Saturday might be different for some systems in regard to week number, but since the crossing is halfway the weekend this is a nice extra for applications that run in companies during office hours. The timestamp will use Wednesday 12:00 as a timestamp, so halfway the week. This latter is to prevent any misunderstandings in case one would like to define a week as beginning on Monday and not on Sunday.

## 2.5 View Content Solver (Vsolver)

The dat

## 2.6 Sgraph Server (Gserver)

The dat

## 2.7 Sgraph Client (Gview)

The dat









## 8. Abbreviations

Abbreviation	Explanation
DCS	Distributed Control System, by i.e. Siemens, Honeywell, Yokogawa, Emerson, Used in chemical and oil & gas industry
MES	Manufacturing Execution System, large plant automation systems, very often based on batch control.
SCADA	Supervisory Control and Data Acquisition, as smaller scope than MES, it can be regarded as a subsystem or predecessor of MES.
DAQ	Data Acquisition System, often as subsystem of DCS, SCADA, MES or ATE.
Real-time	Deterministic software behavior where cycle times are predictive within a defined constraint, used typically where guaranteed precise or fast processing is required.
ATE	Automatic Test Equipment as part of Test & Measurement systems or product test & certification tooling.
WAQ	Sgraph's definition of waveform acquisition method
PAQ	Sgraph's definition of periodic acquisition method
EAQ	Sgraph's definition of periodic acquisition method
FIFO	First-in First-out buffer handling method, common method to prevent data loss for fast acquired signals
cRIO	Compact RIO, a name for an acquisition hardware line as manufactured by National Instruments
PLC	Programmable Logic Controller, Industrial grade control system, Realtime and robust with lots of I/O modules.
SQL	Sequel, a standardized interface to database systems
TDMS	A National Instruments standard file format for storing datasets with high efficiency
BIN	General name for binary files, for data acquisition it can be regarded as "custom built file format"
HDF5	Hierarchical Data Format version 5, a standard for structured metadata storage, very common used by Matlab users.
HMI	Human Machine Interface, another name for Graphical User Interface or GUI
AVPP	Averaging Post Processor, the module that takes care of min/max and averaging.
HISW	Historian Write interface, the high level module that is used to write data.
Gserver	Common set of code that delivers data to the Sgraph viewer
Vsolver	module that determines which data set is needed by the Sgraph
FAMM	Fast Access Memory Manager, the RAM portion of the data historian
RAM	Random Access Memory = the computer's fast onboard working memory.
DBAL	Database Abstraction Layer, the interface between the database and the software queries
Gview	The 'viewer' of the historian database, this is the GUI that is used to view and investigate the historian data
GUI	Graphical User Interface, the part of the software that is used to control and view results, usually by using mouse, keyboard and display. Also more intuitive gesture and touchscreen techniques apply here.
Lvlib	Name for a LabVIEW library, usually used within a LabVIEW project (LabVIEW IDE)
IDE	Integrated Development Environment, central project location for software development like i.e. eclipse Some software development languages can have several IDE platforms, some have one OEM provided platform
OEM	Original Equipment Manufacturer, the original supplier of a product
DVR	Data Value Reference : a LabVIEW datatype to use data 'by reference'
FG	Functional Global : a programming technique to save memory footprint and prevent multiple write race conditions.
Qtype	The sort of acquisition like waveform (WAQ), continuous acquisition (PAQ) or Event based acquisition (EAQ)