Advanced graphical presentation library

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

The objective of ∑graph is to provide a fast way to add modern graphic presentation tools for storing and analyzing large amounts of acquired data.
Sgraph is a leap to turn coding of data viewers into a joyful browsing experience.
Fast ⇒ Intuitive ⇒ Scalable

André Koelewijn, 2020
Beta version V 0.7

1. ∑graph fundamentals 3
1.1 Simplified concept 4
1.2 Data logging narrative 5
1.3 Signal sampling Qtype definitions 7
1.4 Data historian requirements 10
1.5 Computer processing behavior narrative 11
1.6 ∑graph software architecture 12
1.6.1 Software abstraction levels 12
1.7 Dataflow model 13

2. ∑graph library structure overview 14
2.1 DataBase access Abstraction Layer (DBAL) 15
2.1.1 DBAL: writing PAQ files to disk 16
2.1.2 DBAL: PAQ: BIN database format 19
2.1.4 DBAL: PAQ: SQLite database format 24
2.1.5 DBAL: PAQ: TDMS database format 24
2.2 Historian Write Interface (HISW) 25
2.2.1 HISW: Benchmark data source 25
2.2.2 HISW: Writing data to the historian 26
2.2.3 The HISW interface 26
2.3 Fast Access Memory Manager (FAMM) 29
2.3.1 FAMM: AVPP: Anti-Aliasing Layered datasets 29
2.3.2 FAMM: RAM memory: The dataset object 29
2.3.3 FAMM: RAM: TAG data element 30
2.3.4 FAMM: RAM: The TAG Status definition 31
2.3.5 FAMM: RAM: The TAGs memory element 32
2.3.6 FAMM: The PAQ resolution layers 33
2.3.7 FAMM: The PAQ size and display resolution 34
2.3.8 FAMM: The PAQ resolution setting 35
2.3.8 The PAQ data manipulation 36
2.3.9 FAMM: TAG properties 37
2.3.10 FAMM server 38
2.3.11 FAMM initialization 39
2.4 Averaging post processor (AVPP) 41
2.4.1 AVPP: Averaging span 41
2.5 View Content Solver (Vsolver) 43
2.6 ∑graph Server (Gserver) 44
2.6.1 Gserver initialization 44
2.7 ∑graph Client (Gview) 46
2.7.1 Gview basic concepts 47
2.7.2 Gview initialization 48
2.7.3 Gserver Queries 49
2.7.4 Gviewer screen updates 49
2.7.5 Gviewer dedicated settings 49
2.7.6 Gviewer general settings 49

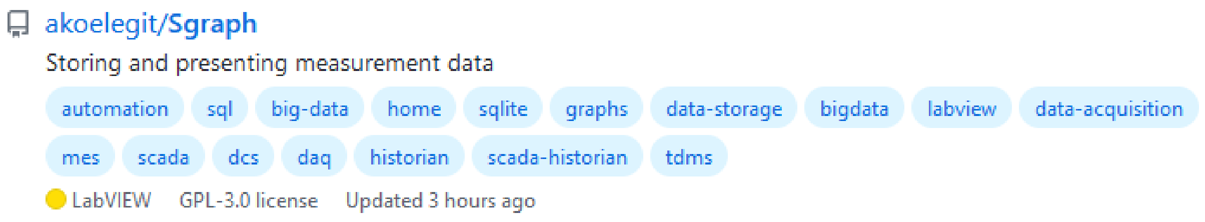
3. Testing ∑graph modules 50
3.1 Test-Sgraph-system.vi 51
3.1.1 Gserver startup 52

8. Abbreviations 53

1. ∑graph fundamentals

The intention of ∑graph is to provide added value for storing repetitive acquired signals fast and easy. At front an investigation is performed to determine if the historian should support waveform type, repetitive clocked, or event based data sources. The ∑graph objective went towards supporting them all.

Author/Developer: Andre Koelewijn, 2020


akoelegit/Sgraph

Storing and presenting measurement data

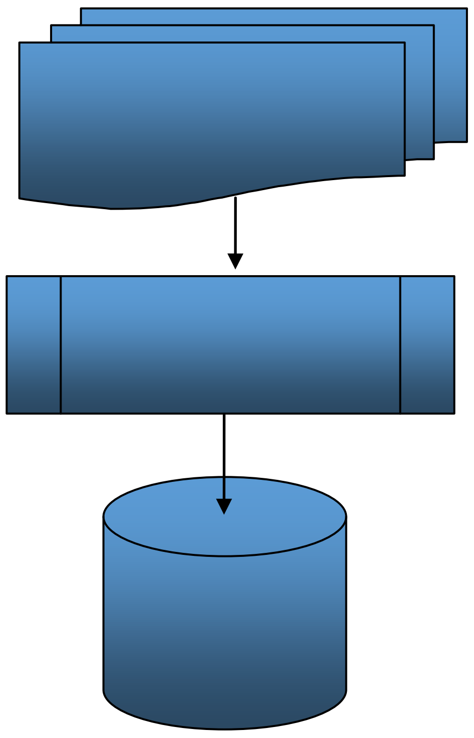
automation sql big-data home sqlite graphs data-storage bigdata labview data-acquisition mes scada dcs daq historian scada-historian tdms

● LabVIEW GPL-3.0 license Updated 3 hours ago

The ∑graph repository can be found on GitHub and is free for users to implement under the GPL-3.0 license. At the moment of writing (April 2022) not all classes are implemented yet, but the first (beta) version works with the standard binary database format and periodic updated datasets (PAQ datasets). All supported datasets are planned to be implemented in 2022, and subsequently the support of the other database classes (SQL, TDMS etc) are planned. Additional database formats might appear depending on requests of users and community contributions.

1.1 Simplified concept

A simplified view on how to build a complete data historian:



Present Acquired data samples

Select how to save (SQL / TDMS / Raw)

Run (and use presentation tools)

This is basically the concept.
Using it should be as simple as that.
For developers and expert users however, there's a little more to grasp...

1.2 Data logging 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 implementation area. Is it a DCS[1] system for a chemical production plant? Is it a test system to perform quality control for industrial production? Is it part of a MES[2] system for large scale industrial SCADA[3]? Is it a stand-alone DAQ[4] or ATE[5] system? Home built solar system? Or professional big data analysis?

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 ∑graph initiative is **not** meant to cover the structural 'application specific' needs. Those are part of the primary design specification. ***∑graph 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…)

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 is actually another approach on troubleshooting where developers often are tempted to process data right after acquisition and the data acquisition is regarded as only a small portion of the project scope. By broadening the project scope by 'first acquire and store everything' and next move to the processing and control tasks (and also store those results if applicable) so one can always look back to initial data in case of a problem.

---

[1] DCS = Distributed Control System, by i.e. Siemens, Honeywell, Yokogawa, Emerson.

[2] MES = Manufacturing Execution System, large plant automation systems, very often based on batch control.

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

[4] DAQ = Data Acquisition System, often as subsystem of DCS, SCADA, MES or ATE.

[5] ATE = Automatic Test Equipment as part of Test & Measurement systems or product test & certification tooling.

A few notes to prevent false assumptions:

Calibration Status
The calibration due dates and calibration settings are not part of the functionality of ∑graph 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 ∑graph. 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.

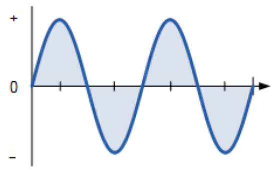
Special functions
The data saving format includes a little reserved space to store custom data states together with the sampled data as 'custom defined status bits'. As the name explains, it's custom, so not covered by ∑graph, just some provided space. So there's some playground space reserved.

1.3 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 ∑graph:

- 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 ∑graph and being the three ***∑graph methods*** to support a vast number of different applications.



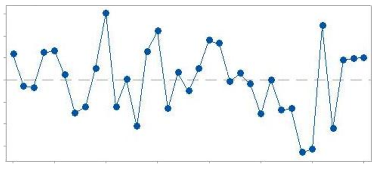
Waveforms (WAQ)

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 finite 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[6] 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.

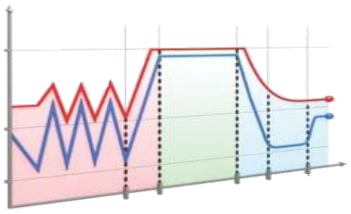
The difference between a waveform and periodic continuous acquisition is the distinction that a waveform makes use of a FIFO to acquire fast sampled data for a limited amount of time.

[6] FIFO = First in, First out, as general concept for a continous acqusition buffer.



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[7] 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[8] system).



Event based sampling (EAQ)

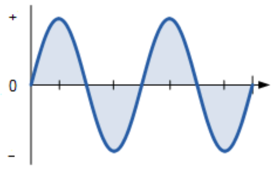
When the several signal sources are not synchronized in such a way that every signal is able to arrive at any random moment in time then event driven sampling should be used. For event signals every single sample has its own unique timestamp. The advantage in opposite of the periodic sampled acquisition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

[7] Distributed Control System, think of a vast number of PID controls in a large chemical plant control system.

[8] National Instruments cRIO chassis with LabVIEW Realtime operating system.

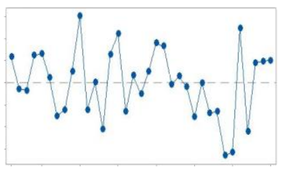
So Just for completeness, let’s list the main properties of each sampling Qtype:

- **WAQ**



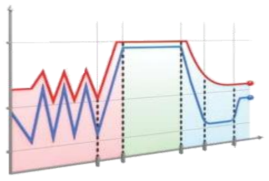
- Very fast acquisition (around 10 ks/sec up to 10 Gs/sec)
- Finite acquisition time (i.e. 10 seconds)
- Timing defined by T_0 and dT (sample speed)
- All channels sampled simultaneous
- Deterministic (usual very accurately timed)
- Typical application field: DAQ[9] and ATE[10] systems

- **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 simultaneous
- Usual non-deterministic (not real time)
- Typical application field: DCS[11] or MES[12] 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

[9] DAQ = Data Acquisition, think of test and measurement systems
[10] ATE = Automatic Test Systems, i.e. test a product functionality by an automated process
[11] Distributed Control System, think of a vast number of PID controls in a large chemical plant control system.
[12] MES = Manufacturing Execution System, large plant automation systems, very often based on batch control.

1.4 Data historian requirements

The objectives of this ∑graph 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 ∑graph 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 ∑graph:

- Sample indefinitely (> 20 years)
- Acceptable database size
- Easy to implement
- Support three Qtypes of acquisition (waveform, periodic, events)
- Several database types (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)

1.5 Computer processing behavior narrative

The computing power and processing focus of ∑graph follow a distinct set of principles to ensure an optimal user experience. To begin with the ∑graph 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[13] interaction should be immediate to give the user an experience of direct responsiveness. Since the AVPP[14] must perform calculations before the average and min/max data is stored, this can cause latency if one is watching this data life or short after acquisition. But still this processing has lower priority then keeping track of the acquisition buffers of the HISW[15] 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[16] where it's submodule Vsolver[17] will determine what dataset is used, and deliver this data with flashing speed to the FAMM[18] 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[19] 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 processing 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 facilitate those 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 with Test & measurement data.

[13] HMI = Human Machine Interface, another name for Graphical User Interface or GUI
[14] AVPP = Averaging Post Processor, the module that takes care of min/max and averaging.
[15] HISW = HIstorian Write interface, the high level module that is used to write data.
[16] Gserver = Common set of code that delivers data to the Sgraph viewer
[17] Vsolver = module that determines which data set is needed by the Sgraph
[18] FAMM = Fast Access Memory Manager, the RAM portion of the data historian
[19] RAM = Random Access Memory = the computer's fast onboard working memory.

1.6 ∑graph software architecture

The ∑graph software makes use of native LabVIEW object oriented software. In other words, classed that use data by value. The ∑graph project did not use any GOOP or other toolkits. There are however locations 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.

1.6.1 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.

∑L0	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.
∑L1	Abstraction Level 1 Now you went one subVI deeper into the system, for experienced programmers no problem, for beginners: be careful here.
∑L2	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.
∑L3	Abstraction Level 3 You’re entering the basic building blocks of ∑graph, don’t change anything here if you don’t understand the design pattern architecture. Expert stuff only.
∑L4	Abstraction Level 4 The foundation blocks of ∑graph. Only expert stuff here. Lots of code and system performance depends on this. No playing around here.
∑L⬇	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.

1.7 Dataflow model

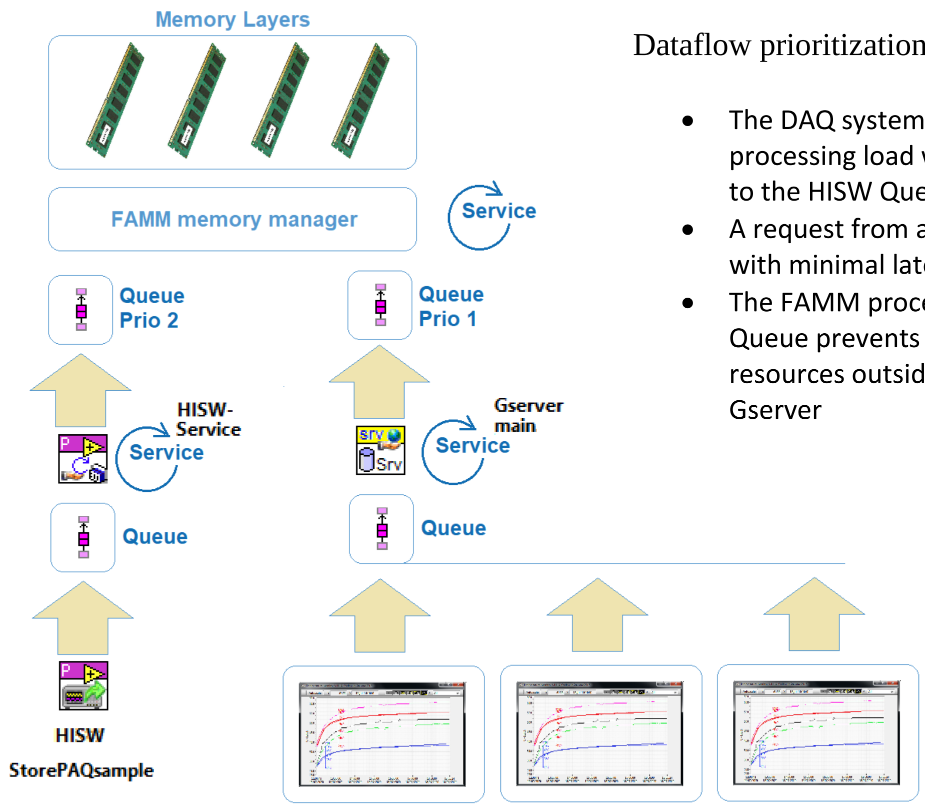
The ∑graph data flow follows the priority definitions of paragraph 1.5:

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

In this list there are three services that operate independent and can potentially run in a separate processing thread:

- ***FAMM***: managing everything that goes to or comes from memory or disk
- ***Gserver***: Process any incoming request from one of the Gviewer displays (the GUI's)
- ***HISW***: process all received samples without a burden to the sending code

In order to let the above three services run independently with maximum usage of the computer's processing power and multithreading potential; the requests to and replies from these services are separated using the LabVIEW Queue services. The LabVIEW Queues can operate and shared between several applications



Dataflow prioritization:

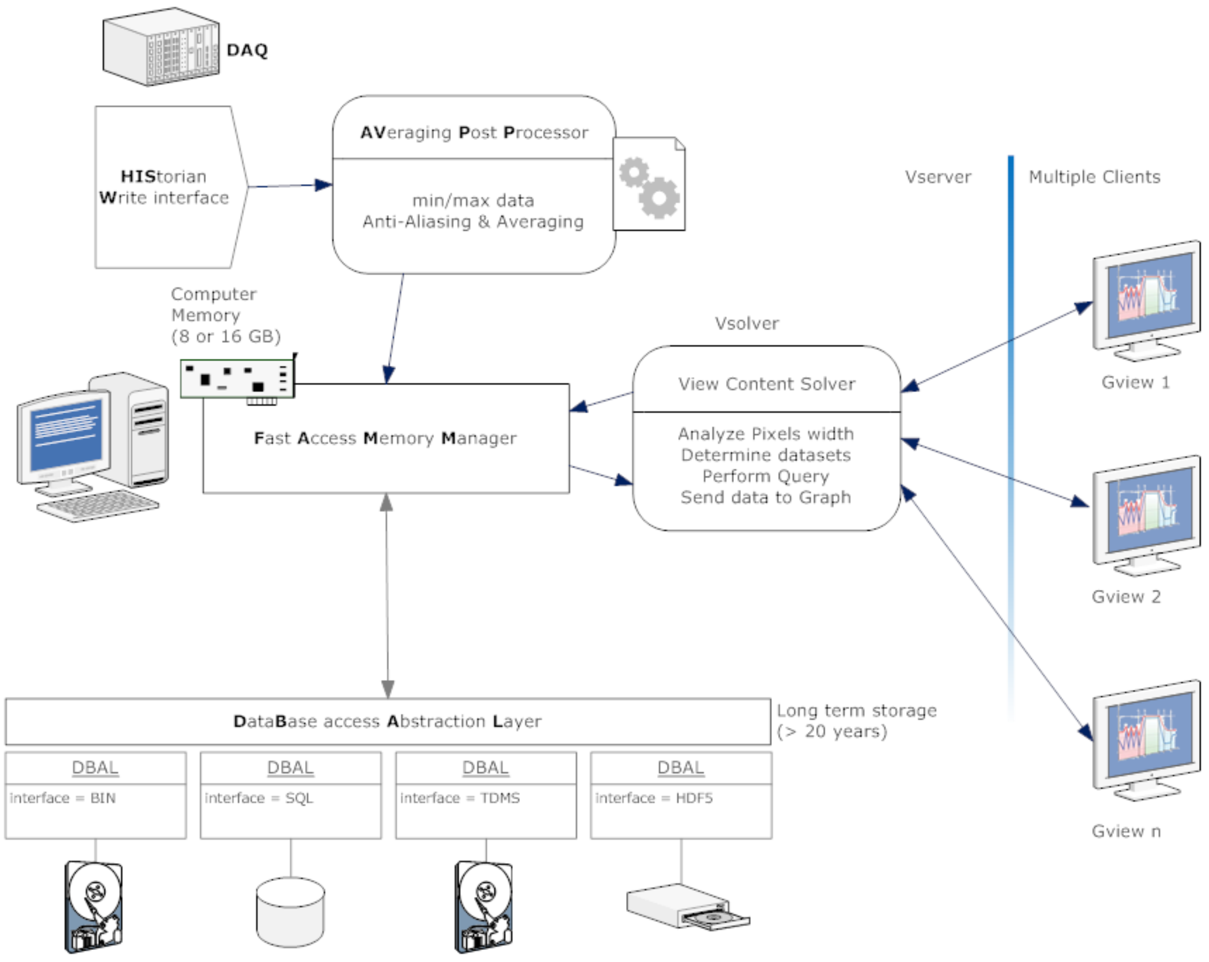
- The DAQ system must experience minimal processing load when storing timestamped data to the HISW Queue
- A request from a Gviewer viewport is processed with minimal latency
- The FAMM processing priorities of the receiving Queue prevents data loss while optimizing CPU resources outside the time critical replies from Gserver

2. ∑graph library structure overview

The ∑graph 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 the experience of 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. ∑graph Server (Gserver)
7. ∑graph 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.



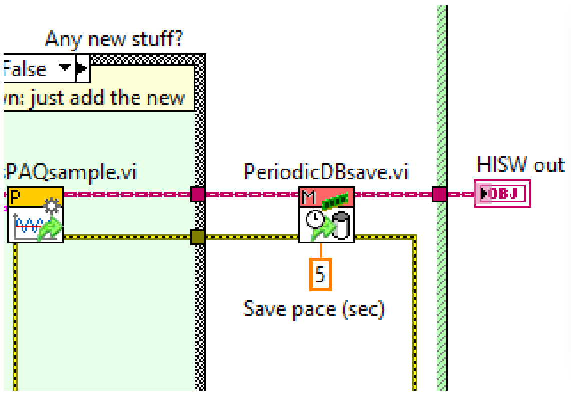
2.1 DataBase access Abstraction Layer (DBAL)

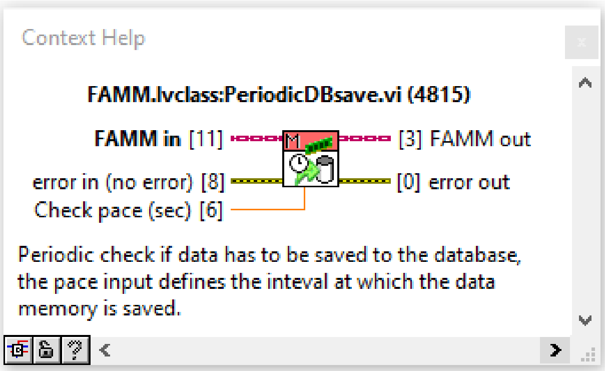
The database abstraction layer plays a role in maintaining a standard interface between data storage on disk and the ∑graph application. Since type of database can vary per designer preference or other circumstances, this abstraction layer or driver makes sure that data storage matches your needs. There might be more needs or other frontends that you would like to use the same data as saved by the ∑graph 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.
- HDF5 optimized for MATLAB / Simulink users

2.1.1 DBAL: writing PAQ files to disk

In order to store the computer memory (RAM) data to disk, a FAMM method called ***PeriodicDBsave.vi*** will store that data to the database.



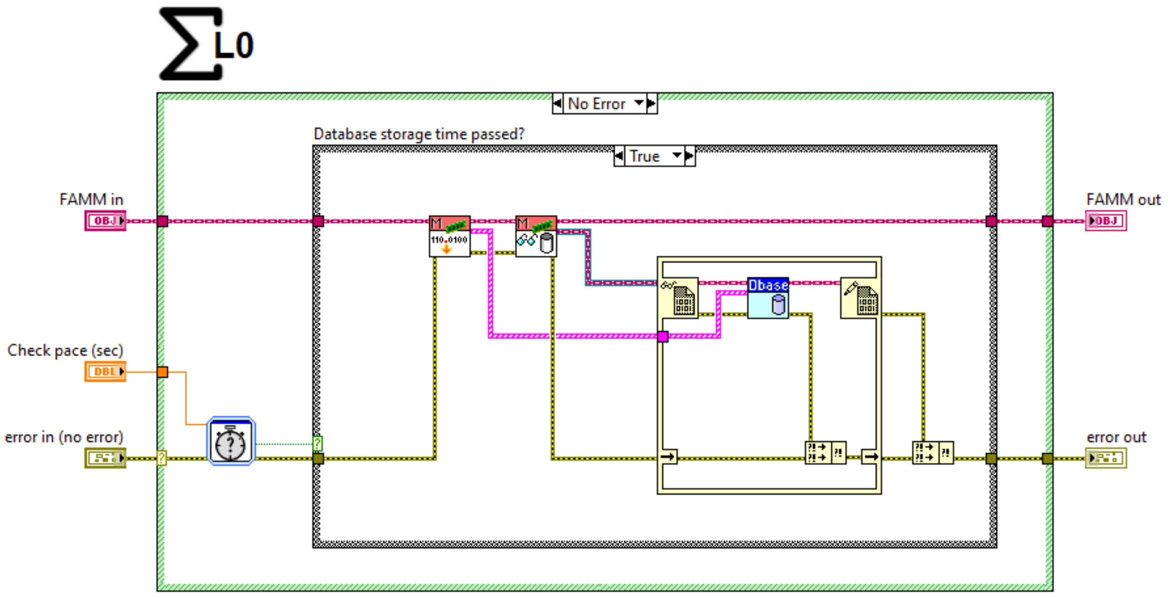


FAMM.lvclass:PeriodicDBsave.vi (4815)

FAMM in [11] — [3] FAMM out
error in (no error) [8] — [0] error out
Check pace (sec) [6]

Periodic check if data has to be saved to the database, the pace input defines the inteval at which the data memory is saved.

In that VI a few FAMM methods will check which kind of database class (BIN, SQL, SQLite, TDMS) is used and first retrieve the latest unsaved portion of the PAQ/WAQ/EAQ memory and then save it using a Memory subclass override VI where an empty parent class exists for any of the Qtypes (PAQ/WAQ/EAQ). Depending on the database type as defined in the ***\∑graph\database\almanac\settings.ini*** file, the respective override class of that type will be used to perform the action. This makes the database maintainable and scalable.



To facilitate the memory pointers that indicate what the latest unsaved portion of the memory was that was not yet saved, a pointer is saved in the DVR based class attribute of every resolution layer. For the PAQ memory; the FAMM class DVR instance contains a PAQ_memory class instance. This memory instance contains a list of TAGs as the only direct accessible data. The actual memory is defined as DVR references of the several resolution layers of the PAQ memory in that same memory instance.
Every one of the resolution layers (L0..L8) instances contain already the database resolution definition (10mil, 100mil …1day, 1wk) name to identify in which database table the data should be saved.

PAQ Layers for several time resolution systems

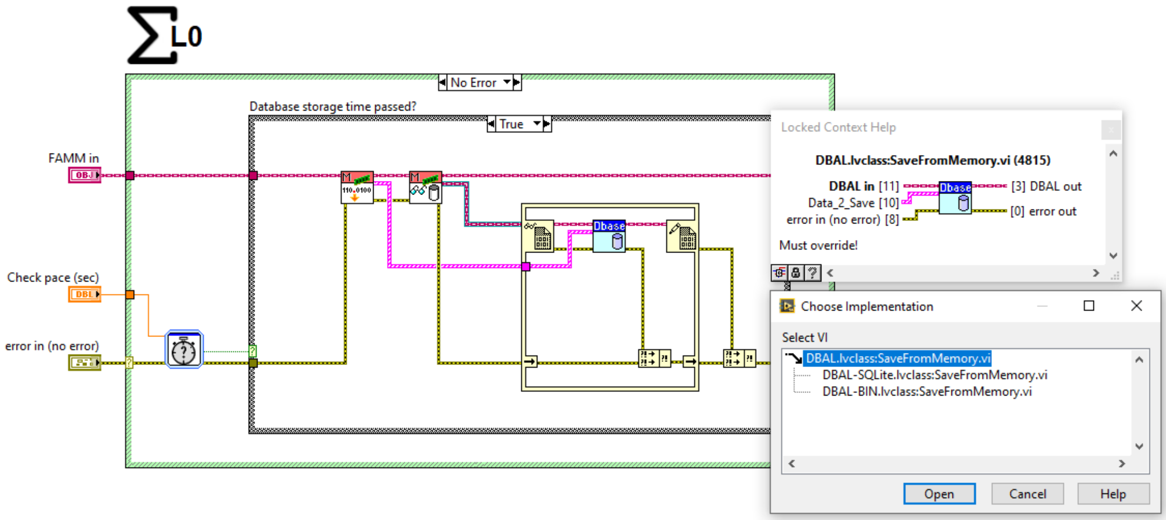
Time resolution of the PAQ data					
10 msec	100 msec	1.0 second	10 seconds		
Layer # as defined per system				Sampling system per layer	Table name
L0				10 millisecond per sample	PAQ_10mil
L1	L0			100 millisecond per sample	PAQ_100mil
L2	L1	L0		1.0 second per sample	PAQ_1sec
L3	L2	L1	L0	10 seconds per 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 Layer definitions (L0 .. L8) are defined by array index, and used by the PAQ memory class. For storing this data into the database; the array index is not important, the database table name (10mil, 100mil …1day, 1wk) will determine in which database table the data is stored.

The difference between the array index (L0, L1 … L7, L8) and the database table name (10mil, 100mil …1day, 1wk) have to do with the universal way the system can be configured. For a system that is configured to store 100 samples per second (10mil), the number of memory elements will differ as compared to a system that is configured to store one sample every 10 seconds (10sec). The number of array elements (resolution depth) will differ, but not the name of the data in the database in respect to the actual time resolution. After all, the database should be self-descriptive for it to be accessible outside the ∑graph primary scope, just in case a developer would like to do so.

The access to the DBAL save routines are by using a database type class. A DBAL class override is needed to access the specific methods that differ per database type.



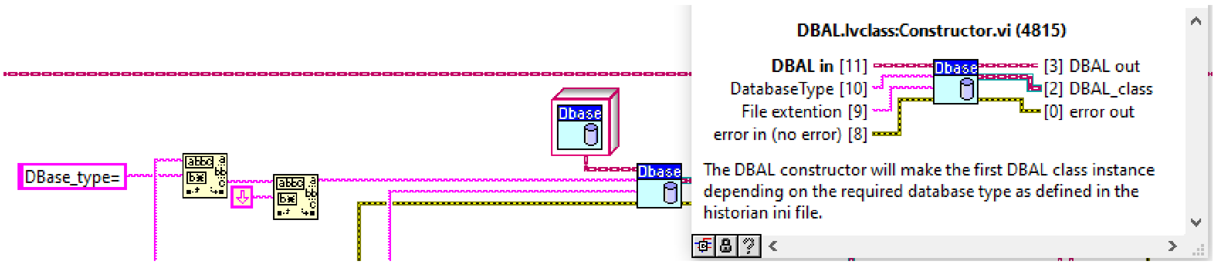
DBAL.lvclass:SaveFromMemory.vi (4815)

DBAL in [11] — [3] DBAL out
Data_2_Save [10]
error in (no error) [8] — [0] error out

Must override!

Select VI
DBAL.lvclass:SaveFromMemory.vi
DBAL-SQLite.lvclass:SaveFromMemory.vi
DBAL-BIN.lvclass:SaveFromMemory.vi

The FAMM class constructor must be called once to create all memory and database instances.
In this FAMM constructor the DBAL constructor VI is called.

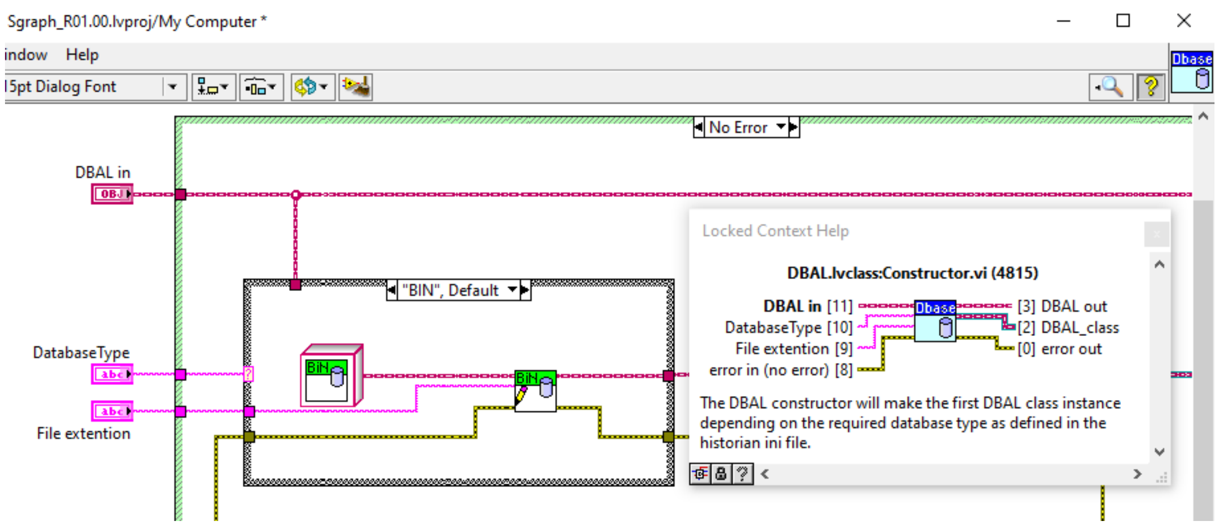


DBAL.lvclass:Constructor.vi (4815)

DBAL in [11] — [3] DBAL out
DatabaseType [10] — [2] DBAL_class
File extention [9] — [0] error out
error in (no error) [8]

The DBAL constructor will make the first DBAL class instance depending on the required database type as defined in the historian ini file.

In this initialization the database dependent DBAL class is instantiated.
From there on, the correct class is automatically utilized.



DBAL.lvclass:Constructor.vi (4815)

DBAL in [11] — [3] DBAL out
DatabaseType [10] — [2] DBAL_class
File extention [9] — [0] error out
error in (no error) [8]

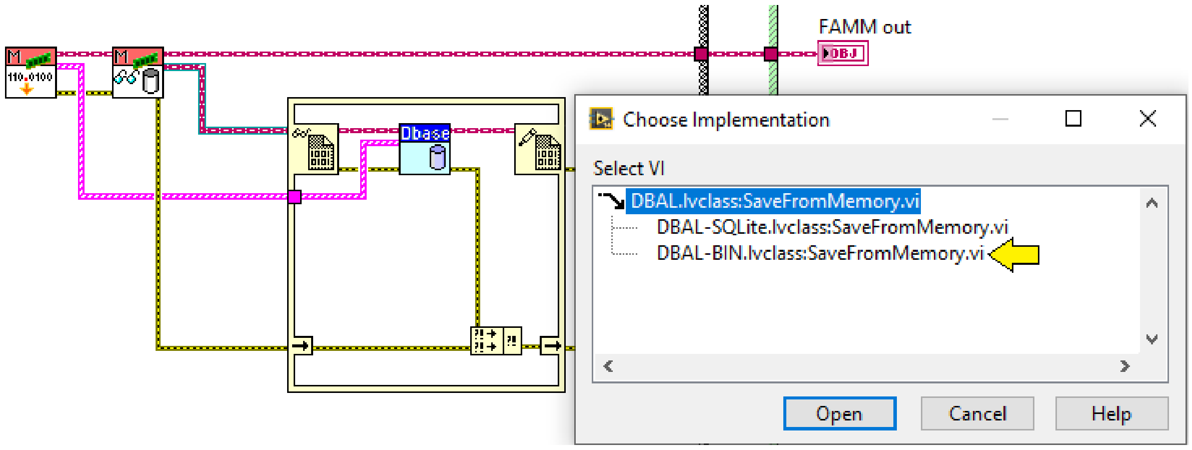
The DBAL constructor will make the first DBAL class instance depending on the required database type as defined in the historian ini file.

2.1.2 DBAL: PAQ: BIN database format

The binary database format (BIN) is where it all started. Predecessors of this software used a custom file format where the header info was stored in ASCII and the data content in compact but uncompressed binary format. The reason to use uncompressed data is to prevent any additional latency where the combination of ASCII header and binary content was a compromise between efficiency and readability. At that time I did some research what file format to use, discussed the approach and acted on what seemed the best solution at that time. It was interesting to see that later other people tried to modify the software based on evolving insight (or just another team leader that would like to have a personal touch). Whatever the reason can be: it showed me that software should at best be adaptable with more modular design so it does not depend on personal preference, evolving insight, shifts in technology or efficiency for specific application properties.

The BIN file format aims at the following specific properties:

- Not depending on TDMS, since this format had a history of a few annoying bugs.
- Not depending on additional libraries like SQLite, SQL or other.
- Fast access due to binary data file area.
- Save memory space in respect to pure ASCII data.
- Open code down to a fundamental level of file access VI's



DBAL.lvclass:SaveFromMemory.vi
DBAL-SQLite.lvclass:SaveFromMemory.vi
DBAL-BIN.lvclass:SaveFromMemory.vi

Once the DBAL-BIN class is selected then the override classes will appear that contain the BIN database type. The BIN class makes use of plain file access to save the historian data as flattened data format.

The historian data files are stored in the ***\Sgraph\database\historian*** file location where the ***\Sgraph\database\almanac*** folder contains the ***settings.ini*** file where the binary file extensions are defined.

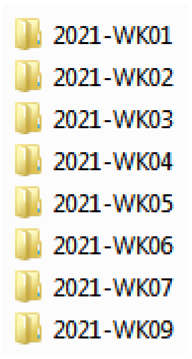
The ***Almanac*** subfolder contains all database settings and TAG definition settings.
The ***database*** subfolder contains all actual database.

2.1.2.1 DBAL: PAQ: BIN database layer size

The BIN file tables are sized with a certain maximum, this is currently set to 32 MByte. If a file reaches this limit, it will create a new file with a new sub sequential numerical value. The filenames will start with a timestamp value for it's filename followed by the sequence number. To limit the number of files per folder; the database contains a number of subfolders per time period that depends upon the layer used.

A folder with notation <YYYY>_WK<ww> (year and week) is used for the layers:

- PAQ_10mil
- PAQ_100mil
- PAQ_1sec
- PAQ_10sec



2021-WK01
2021-WK02
2021-WK03
2021-WK04
2021-WK05
2021-WK06
2021-WK07
2021-WK09

These file have the format YYYY_MM_DD_nnn
where nnn represents the sequence number.
Whenever the file content will cross a Day then a new filename is created.

The layers:

- PAQ_1min
- PAQ_10min

Remain in the year folders and have a week notation per file as per
YYYY_WW_nnn
where the nnn is a seqential order number.

The Layers:

- PAQ_2hr
- PAQ_1day
- PAQ_1wk

Remain in the WIDE folder with the notation per file as per
YYYY_nnn
where the nnn is a sequential order number.

For all the above files the number of sequential file segments depends upon the number of PAQ TAGs stored. The next table shows how many files can be expected when a maximum file size of 32 Mbyte is maintained for the examples of 100 and 1000 TAGs.

Layer	# files @ PAQ100 tags	# files @ 1000 PAQ tags	Folder
PAQ_10mil	288	2880	YEAR_WK
PAQ_100mil	29	290	YEAR_WK
PAQ_1sec	3	29	YEAR_WK
PAQ_10sec	1	3	YEAR_WK
PAQ_1min	1	4	YEAR
PAQ_10min	1	1	YEAR
PAQ_2hr	18	175	WIDE
PAQ_1day	2	15	WIDE
PAQ_1wk	1	2	WIDE

Large file counts are therefore only expected in the YEAR_WK folders.

2.1.2.2 DBAL: PAQ: BIN data segmentation

When starting this project using the BIN data format, the choice was made to allow saving chunks of data as 'data segments' where at readback those data segments were stiched together again to a complete data set (depending on how much data was needed to read back from that file). However, the data stitching consumes processing power and thus: time. The segmentation does save time during saving and therefore lowers processing time during continuous PAQ data on periodic intervals (at this time of development set to 5 seconds pace). During system startup I found that the 'data stitching' during file read did cost much time, up to a minute or so. This was regarded as unacceptable. If that time is also consumed during read back of older saved data records then the whole effect of 'flashing fast read back' is not met. It is possible to allow for segmentation with a maximum, but I thought it would be better just to consume more computing power during every file save event (currently set to 5 seconds) and do a complete 'Read / Add / Write' action to replace the previous 'Write segment' routine.

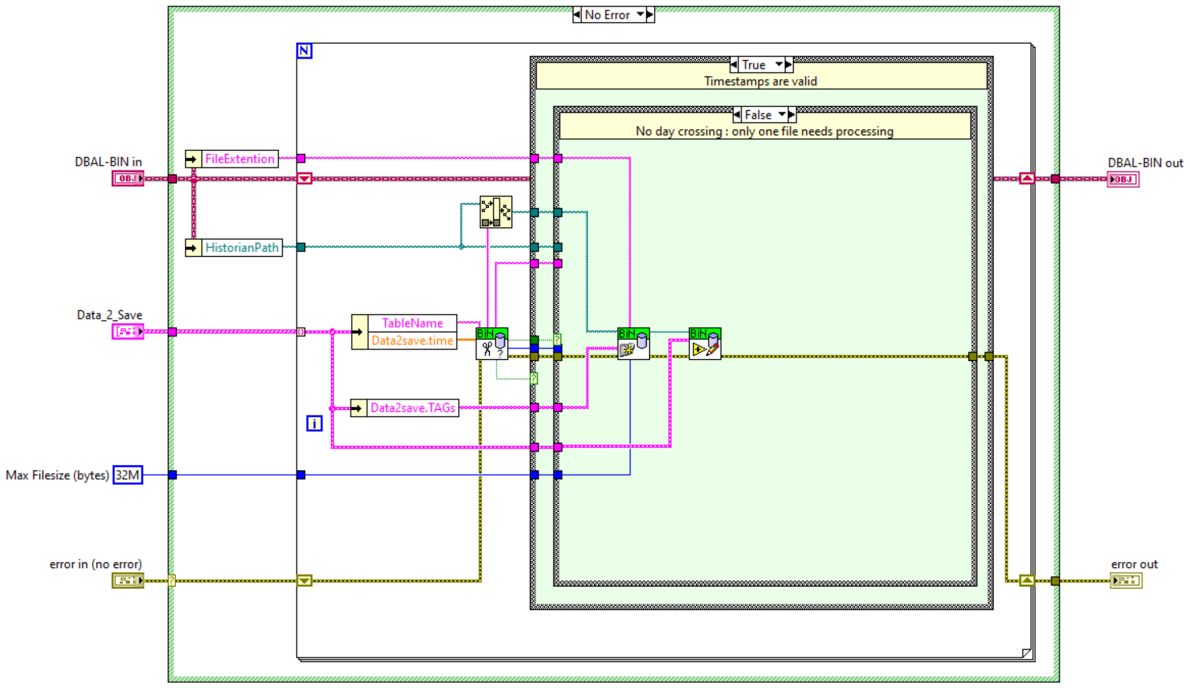
So the choice is made: 'NO SEGMENTATION' and accept the increase in continuous computing power to improve the flashing high speeds at reading, since that fast response is the basis of the S-graph intended look & feel.
And as a result: no defragmentation needed anywhere.

The code is adapted to support that in May 2021.

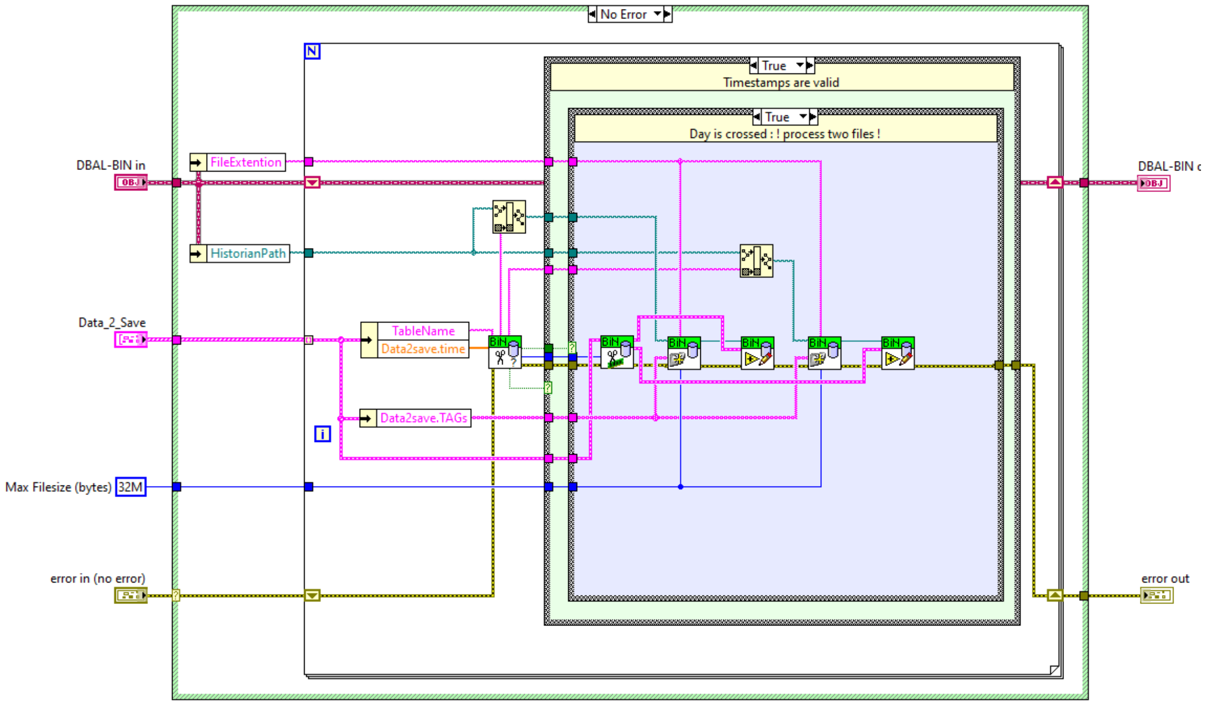
2.1.2.3 DBAL: PAQ: File Indexing

When many PAQ files are stored, the reading software must be able to decide which of those files to use based on start/stop time. Reading the file content to determine that can be time consuming and is therefore not acceptable. A better way would be to keep track of the file content by reading and writing to an index. The fastest way to use this index without keeping track of that on forehand is using an index file. Such an index file is therefore implemented starting June 2021 and saved under the same name and in the same folder as the PAQ files itself, but using file notation:
<date info>_PAQ_IDX_<tablename>.ext
The index file is updated together with saving the PAQ files so it will allways be up to date. A readback action will then first read the index file, determine the sequence file need and fetch the actual data. Since the index file is relative small this does not add significant fetch time to the read action.

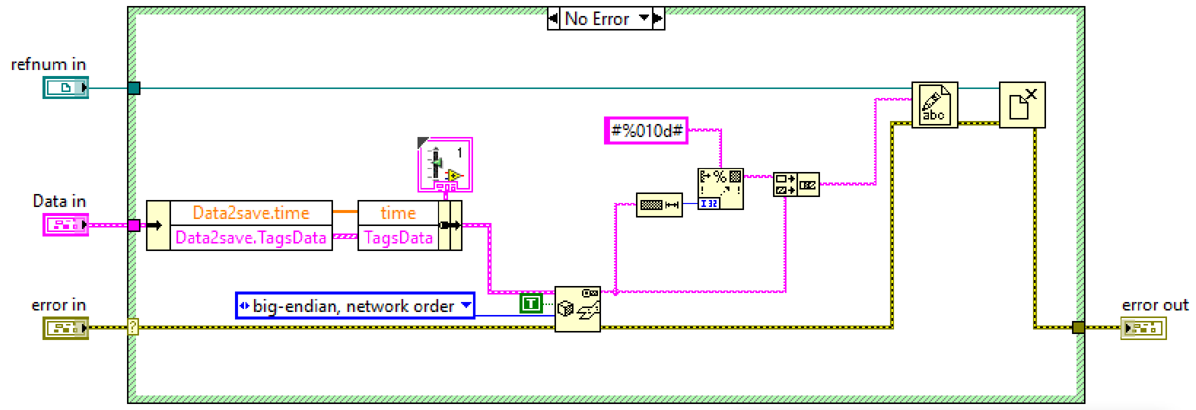
If the DBAL BIN class is accessed to save the FAMM content then a first check is performed to verify if the data can be written into one database record or if it has to be saved into two subsequent records. This can occur if after 23:59 hours one day is passed, where at least one BIN record per day is used.

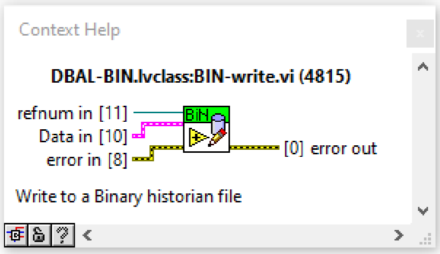


If required so, then the FAMM content will be saved using two separate files.



The actual saving of the BIN data to the appropriate database record is performed using a standard flattening function from the standard LabVIEW palette.



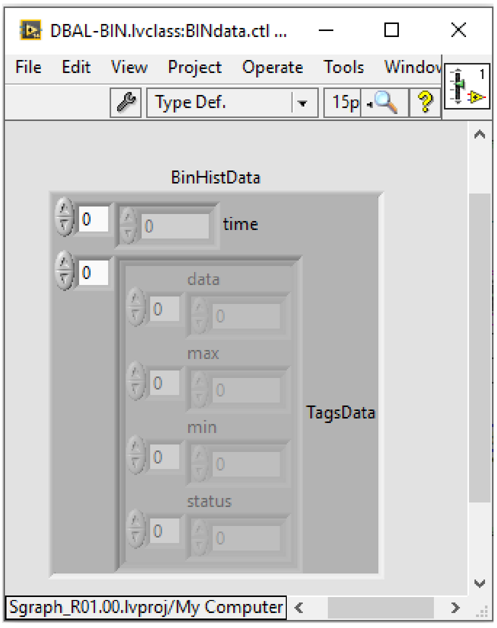


DBAL-BIN.lvclass:BIN-write.vi (4815)

refnum in [11]
Data in [10]
error in [8]
[0] error out

Write to a Binary historian file

The data is formatted using the data types as stored in the BIN class controls folder.



The ***BinHistData*** format is tailored to support the saving of FAMM class records and is used for both flattening as unflattening the binary data records.

The BIN data and FAMM memory DVR memory format is equal for every PAQ resolution layer.

One time array containing as much elements as there are samples in this record.

The TagsData array is sized according the number of saved TAGs in the BIN record header. Every TAG element contains 4 arrays, being data/max/min/status information where every array is sized equal as the time array. Every time array elements belongs the same element of every data/max/min/status array.

Note: it is possible to change the content and TAG order of data to be saved. The DBAL class will check that and make a new record if the TAG list is appended with new data.

2.1.4 DBAL: PAQ: SQLite database format

2.1.5 DBAL: PAQ: TDMS database format

2.2 Historian Write Interface (HISW)

The universal input interface for historian ∑graph 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 ∑graph data historian.

2.2.1 HISW: Benchmark data source

In order to develop and test the ∑graph 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 ∑graph 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.

Historian disk usage:

On may 2020 a test is performed with a BIN database format and 100 samples per second for 100 TAGs. This lead to a database footprint of 1 GB/hour, 25 GB/day, 1TB/year.

2.2.2 HISW: 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.

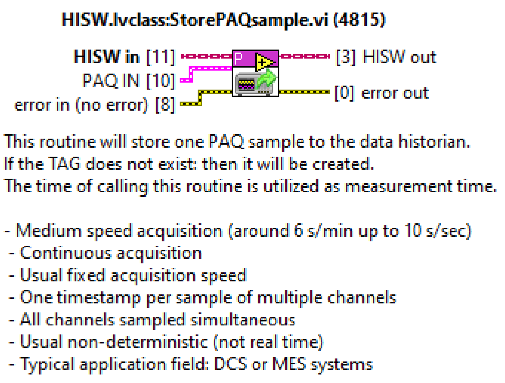
The user does not have to mind about the configuration of the database and it's TAG's. When writing data: the HISW will check if all TAG's exist. If not: then it will extend the existing database with the TAG's it sends. This 'auto-grow' mechanism takes more time than the design latency of the system, 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.

2.2.3 The HISW interface

Writing periodic sampled data (PAQ) to the historian is performed by utilizing the StorePAQsample.vi

This HISW write routine will detect to which resolution layer to write by the settings that are fetched during initialization.



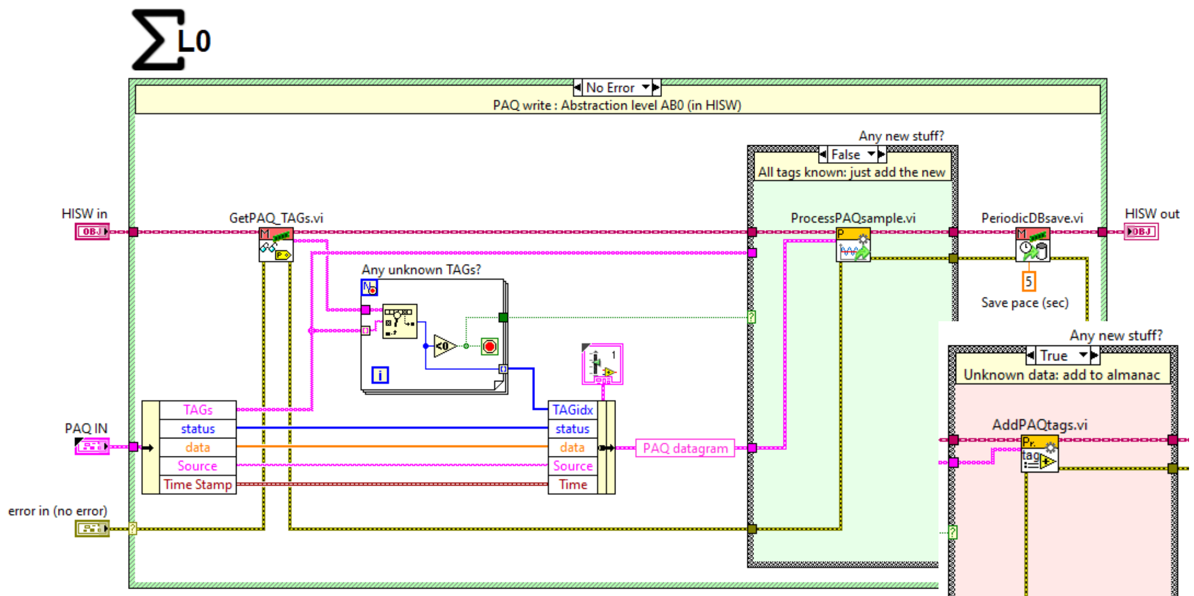
HISW.lvclass:StorePAQsample.vi (4815)

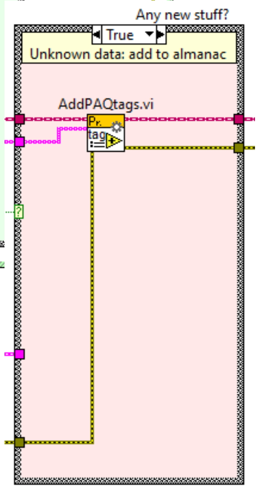
HISW in [11] → [3] HISW out
PAQ IN [10]
error in (no error) [8] → [0] error out

This routine will store one PAQ sample to the data historian.
If the TAG does not exist: then it will be created.
The time of calling this routine is utilized as measurement time.

- 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 simultaneous
- Usual non-deterministic (not real time)
- Typical application field: DCS or MES systems

This HISW routine will first fetch the existing known TAGs of the system and then compare those with the number of TAGs applied at the input. If the number of supplied TAGs are not complete, then those will be updated using NaN data. If more TAGs are applied than present in the TAG definitions then the additional TAGs will be automatically added to the list of defined TAGs including the information that is was HISW that added the TAGs. In this way it can always be identified which source modified the TAG list.



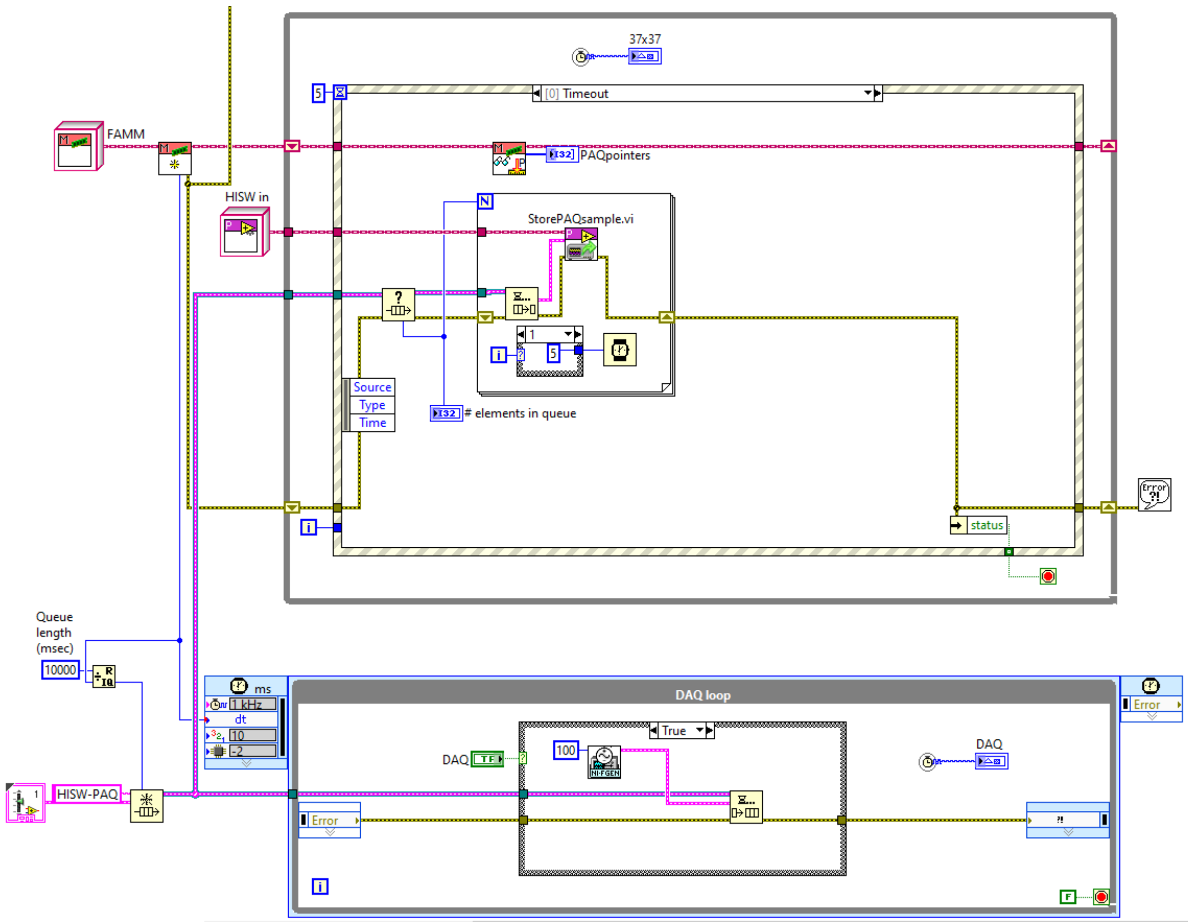


The green case status shows the ***ProcessPAQsamples.vi*** which will update tags if the TAG list is complete, the red case status shows the ***AddPAQtags.vi*** that will add TAGs to the PAQ tagl ist if they do not exist.

On the right side the ***PeriodicDBsave.vi*** will check if there is new unsaved FAMM data in the buffer, and it will save it using the DBAL driver.

The VI’s in this HISQ interface for PAQ samples are of the AVPP and FAMM classes.

A convenient way to write data into the HISQ interface is using a Queue. By utilizing a queue it is possible to make an independent running data acquisition loop while processing the data at a more convenient time. Such a producer-consumer design pattern can balance the processor load while ensuring the data acquisition timing accuracy.



It is not strictly necessary to utilize the design pattern example as shown above. Many different ways of programming are allowed, as long as the main processing VI's are run periodically:

- Fill the HISW interface with valid data
- Ensure that data is periodically saved to disk
- Ensure that the Gviewer server VI runs

2.3 Fast Access Memory Manager (FAMM)

The Fast Access Memory Manager (FAMM) is the RAM memory manager of ∑graph. 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 FAMM: AVPP: Anti-Aliasing Layered datasets

One of the key building blocks of ∑graph is the used of layered datasets. To create fast responsiveness of the Gview displays if large dataset are requested; the ∑graph 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 ani-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 FAMM: RAM memory: 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.

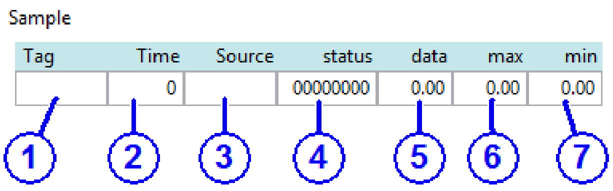
2.3.3 FAMM: RAM: TAG data element

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 ∑graph, 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



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***

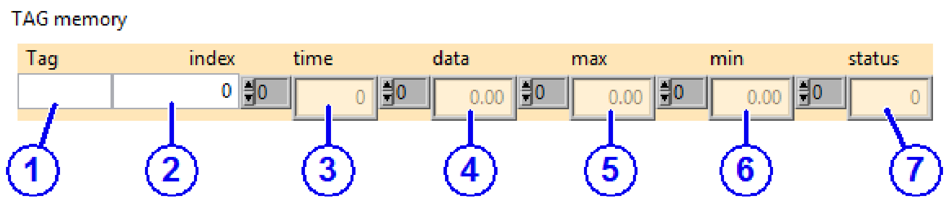
2.3.4 FAMM: RAM: The TAG Status definition

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 FAMM: RAM: 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 FAMM: 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 FAMM: 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 ∑graph 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 for memory set as 'Largest' or '8k'

Layer	Samples	Memory	Timespan max memory	Timespan HD display
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		15 Mbyte	(per TAG)	

A more detailed test showed 14.1 Mbyte for all layers per tag.
For above examples one should realize that a full HD graphical presentation is already extremely high. Usually a presentation having 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 and future proof.

The 32k per layer size can be set to 'Small' (4k) and 'Normal' (8k) and 'Large' (32k) in the ini file setting under parameter 'MemorySize ='

Available timespan and memory footprint per layer for memory set as 'Large' or '4k'

Layer	Samples	Memory	Timespan max memory	Timespan HD display
PAQ_10mil	16k	1 Mbyte	2.7 minutes	19.2 seconds
PAQ_100mil	16k	1 Mbyte	27 minutes	3.2 minutes
PAQ_1sec	16k	1 Mbyte	4.5 hours	18 minutes
PAQ_10sec	16k	1 Mbyte	1.8 days	3 hours
PAQ_1min	16k	1 Mbyte	11 days	18 hours
PAQ_10min	16k	1 Mbyte	16 weeks	1 week
PAQ_2hr	16k	1 Mbyte	3.5 years	3 months
PAQ_1day	4k	256 kbyte	11 years	3 years
PAQ_1wk	1k	low	20 years	21 years
Total		7,5 Mbyte	(per TAG)	

Available timespan and memory footprint per layer for memory set as 'Normal' or '2k'

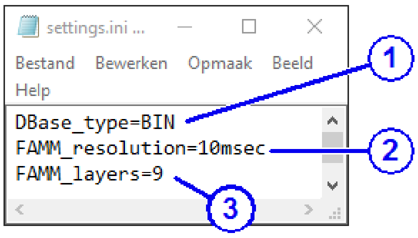
Layer	Samples	Memory	Timespan max memory	Timespan HD display
PAQ_10mil	8k	512 kbyte	80 seconds	19.2 seconds
PAQ_100mil	8k	512 kbyte	13 minutes	3.2 minutes
PAQ_1sec	8k	512 kbyte	2.2 hours	18 minutes
PAQ_10sec	8k	512 kbyte	22 hours	3 hours
PAQ_1min	8k	512 kbyte	5.5 days	18 hours
PAQ_10min	8k	512 kbyte	8 weeks	1 week
PAQ_2hr	8k	512 kbyte	22 months	3 months
PAQ_1day	2k	128 kbyte	5.5 years	3 years
PAQ_1wk	1k	low	20 years	21 years
Total		4 Mbyte	(per TAG)	

Available timespan and memory footprint per layer for memory set as 'Small' or 'HD'

Layer	Samples	Memory	Timespan max memory	Timespan HD display
PAQ_10mil	4k	256 kbyte	40 seconds	19.2 seconds
PAQ_100mil	4k	256 kbyte	6.7 minutes	3.2 minutes
PAQ_1sec	4k	256 kbyte	67 minutes	18 minutes
PAQ_10sec	4k	256 kbyte	11 hours	3 hours
PAQ_1min	4k	256 kbyte	2.8 days	18 hours
PAQ_10min	4k	256 kbyte	4 weeks	1 week
PAQ_2hr	4k	256 kbyte	11 months	3 months
PAQ_1day	2k	128 kbyte	5.5 years	3 years
PAQ_1wk	1k	low	20 years	21 years
Total		2 Mbyte	(per TAG)	

2.3.8 FAMM: 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

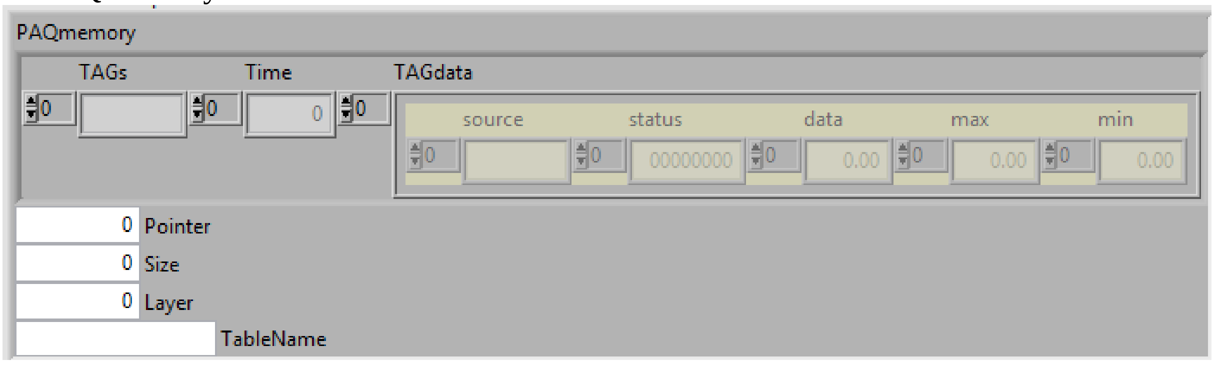


```
settings.ini
DBase_type=BIN
FAMM_resolution=10msec
FAMM_layers=9
```

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.

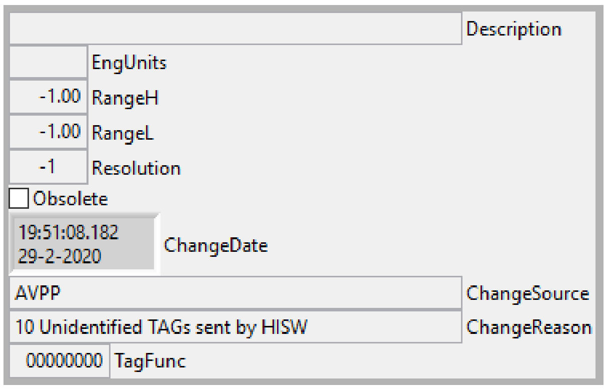
The PAQmemory shows that the TAGs are independent of TAGdata to empower fast search over the 1D TAGs array for read/write actions per TAG id.
One might observe that there is only one [Time] array. Please realize that the [Time] array does NOT equal the [TAGs] Array, but is equal in size of every array size of one of the elements that reside within one [TAGdata] element. In example; the 33rd element of TAGs contains the name of the content of the 33rd element of TAGdata. If someone would like to know the data of the 200th element of the buffer, then the 200th element of all elements of the 33rd element of [TAGdata] should be fetched. For the timestamp of every one of that 33rd element of [TAGdata], the corresponding 200th element of [Time] should also be retrieved.

2.3.9 FAMM: 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



	Description
	EngUnits
-1.00	RangeH
-1.00	RangeL
-1	Resolution
☐	Obsolete
19:51:08.182 29-2-2020	ChangeDate
AVPP	ChangeSource
10 Unidentified TAGs sent by HISW	ChangeReason
00000000	TagFunc

Typedef:

\Sgraph\source\DBAL
\controls\DBAL_TAGpropEdit.ctl

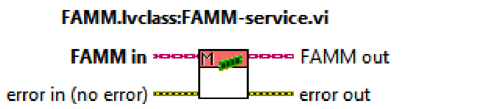
\Sgraph\source\Common
\controls\typedef_com_TagProp.ctl

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 ∑graph, 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.

2.3.10 FAMM server

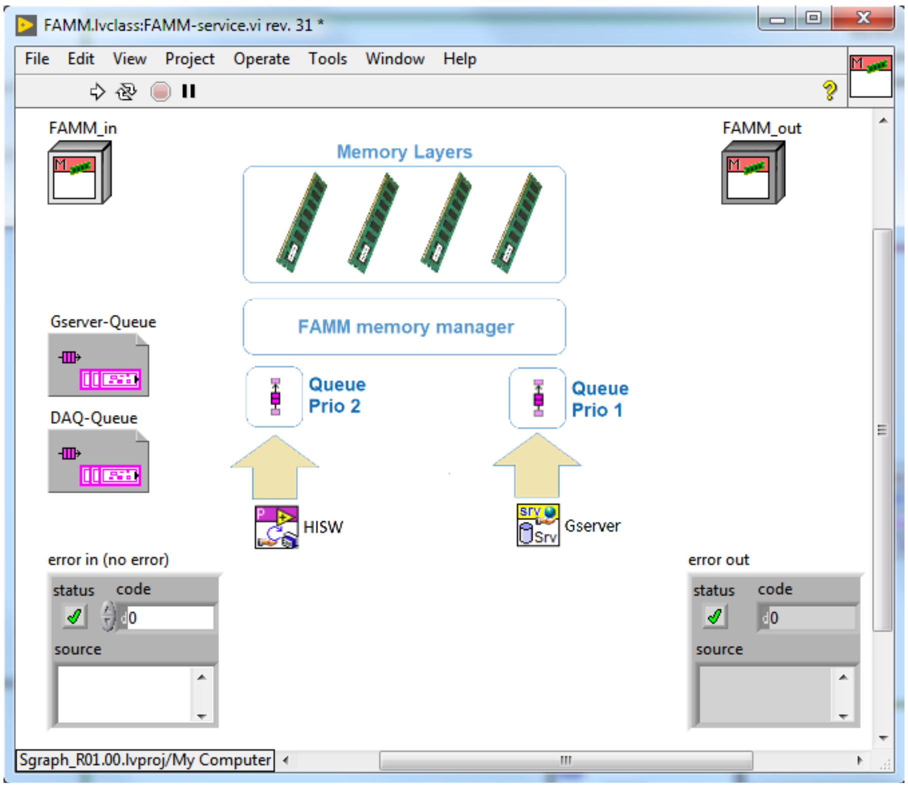
The FAMM Service is started during startup of the system by the Gserver constructor. When the Gserver constructor creates an instance of ***FAMM-service.vi***



The FAMM server is instantiated by Gserver
The FAMM server will process all queue driven commands to read and write from memory, and via the FAMM server access memory or database content.

Whether RAM or Database content is accessed is decided by the FAMM service. Database content is not intended to be accessed directly without using the FAMM service.

Lower level processes to write to the FAMM via the AVPP and HISW service are initiated by the constructor of the FAMM service.

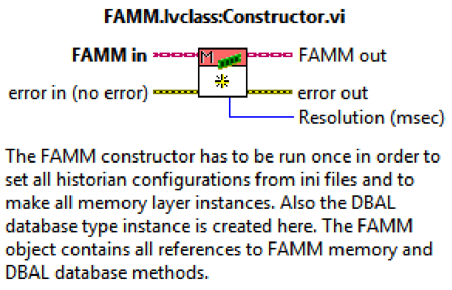


When considering the functionality of the FAMM service, it is important first to understand 'who will start the FAMM service' because the FAMM service is started by the Gserver application. During system startup the Gserver will open a FAMM instance and first load the Gserver-Queue and DAQ-Queue controls with the references of those Queues that were earlier obtained. So at the moment of start; the FAMM service already has valid references of those Queues available. There is a chain of actions needed for total system startup and starting up the FAMM service is one of them.

Since the FAMM service will create and maintain the RAM memory blocks; it should be considered as the most important service for providing fast data access for both reading and writing.

2.3.11 FAMM initialization

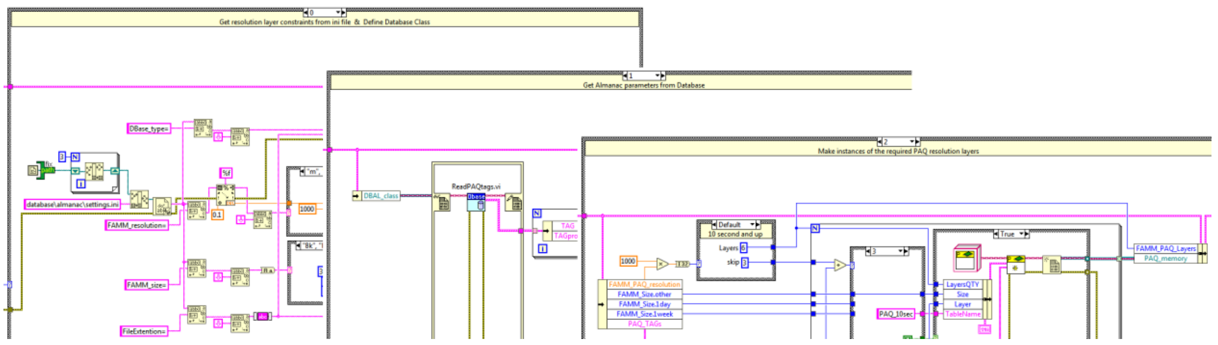
When the FAMM service is started then the first subVI to run is the FAMM constructor.



FAMM.lvclass:Constructor.vi

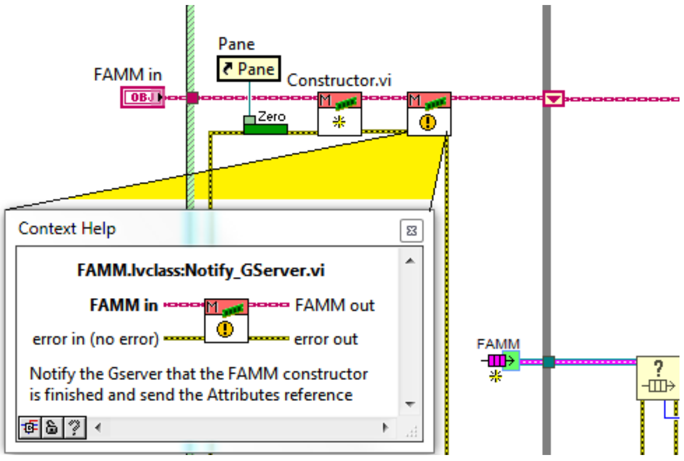
The FAMM constructor has to be run once in order to set all historian configurations from ini files and to make all memory layer instances. Also the DBAL database type instance is created here. The FAMM object contains all references to FAMM memory and DBAL database methods.

This FAMM constructor will first use database ini files to determine how to setup the RAM memory layers and then will try to fetch existing data from the database if present.


Get resolution layer constraints from ini file & Define Database Class
Get Almanac parameters from Database
Make instances of the required PAQ resolution layers
DBase_type=
database\almanac\settings.ini
FAMM_resolution=
FAMM_size=
FileExtention=
DBAL_class
ReadPAQtags.vi
FAMM_PAQ_Layers
PAQ_memory

The RAM memory is built on a number of DVR containers that contain all memory layers in a way that only one memory copy of the data exists which can only be accessed by the private methods of the FAMM service.

When the FAMM constructor is finished then a subsequent VI is called that will notify the Gserver that the initialization phase of the FAMM service is finished, meaning that the FAMM service is operational:



FAMM.lvclass:Notify_GServer.vi

FAMM in — FAMM out

error in (no error) — error out

Notify the Gserver that the FAMM constructor is finished and send the Attributes reference

This step is needed to let the Gserver finish subsequent initialization steps that are needed when the FAMM service is operational.

Additionally the FAMM queue reference is obtained so messages can be processed by the FAMM server.

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 it's 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 AVPP: Averaging span

The resolution layer time slices are defined according 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 a 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 ad 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.

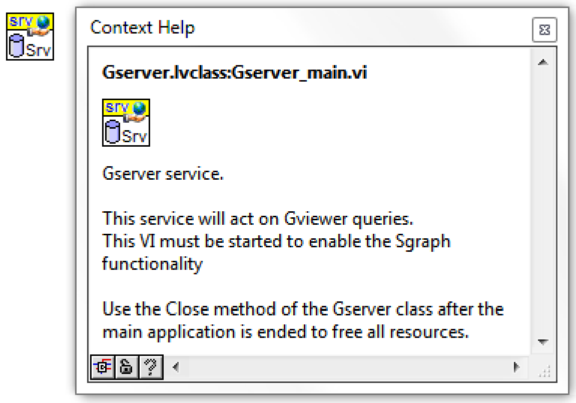
For the PAQ_1wk the week definition of Sundays.. Saturday might be different for some systems in regard to week number, but since the crossing is halfway the weekend this is a nice extra for applications that run in companies during office hours. The timestamp will use Wednesday 12:00 as a timestamp, 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 ∑graph Server (Gserver)

The Gserver main server should run somewhere in your main program where it can run continuously and maintain all services.



Gserver.lvclass:Gserver_main.vi

Gserver service.

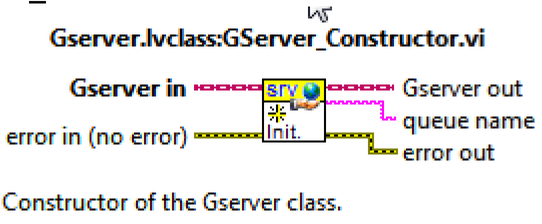
This service will act on Gviewer queries.
This VI must be started to enable the Sgraph functionality

Use the Close method of the Gserver class after the main application is ended to free all resources.

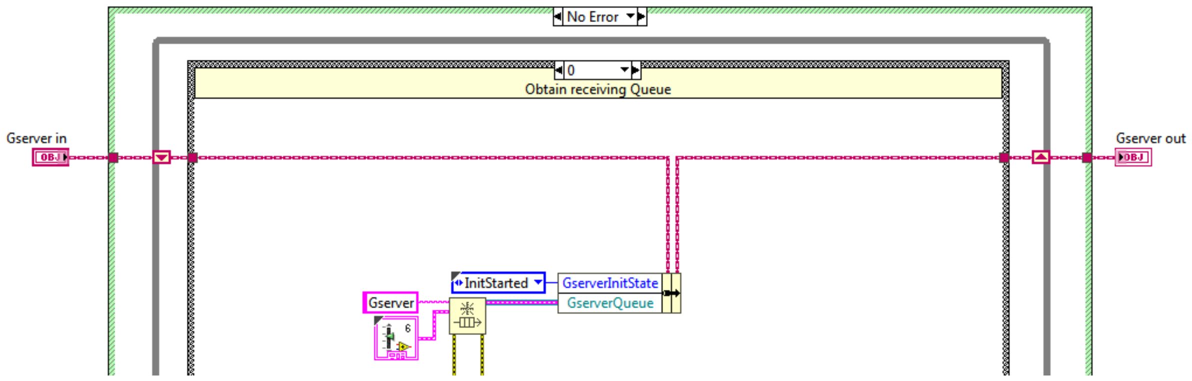
Gserver is the main core VI that will serve all other services using queued messges.

2.6.1 Gserver initialization

When the ***Gserver_main.vi*** is called then it will start with an initialization routine:

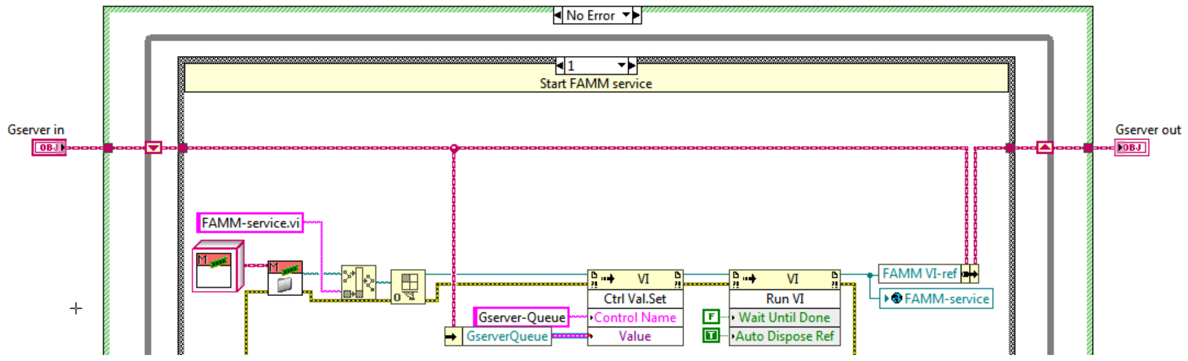

Gserver.lvclass:GServer_Constructor.vi
Gserver in
error in (no error)
srv
Init.
Gserver out
queue name
error out
Constructor of the Gserver class.

As first action the constructor will start the Gserver Queue:



This Queue is needed to process all requests to the server for Gviewer data queries to the memory or database.

As subsequent action of the Gserver constructor it will start the memory manager application by dynamically loading and running the FAMM service:

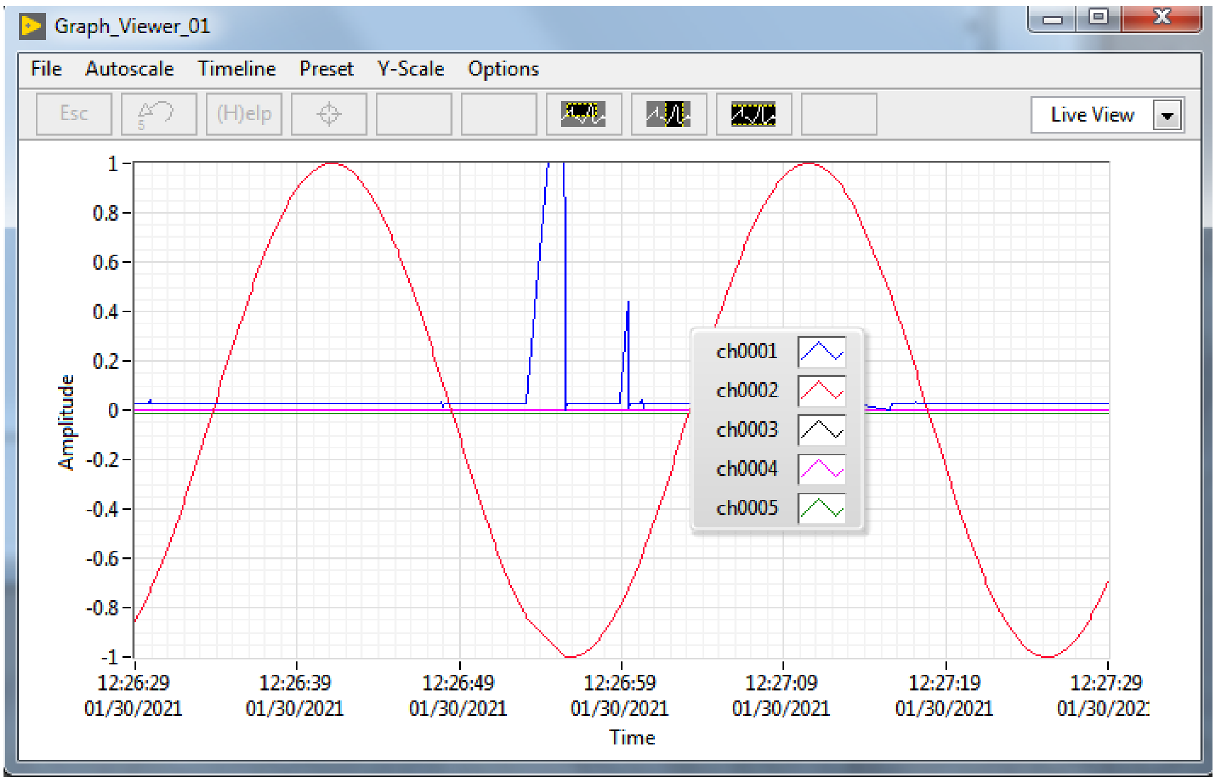


The Gserver Queue reference is written to the FAMM service as a control value so the FAMM service can immediately respond that the startup has succeeded by means of the Gserver Queue.

If the FAMM service constructor is finished with building up the memory blocks and retrieving existing data from the database to pre-fill the memory with existing data, then it will respond back to the Gserver queue by sending the command: ***"SetFAMMobject"*** which will in fact initiate the next phase of Gserver initialization:

2.7 ∑graph Client (Gview)

The part of the program that will show the stored data to the user is the Gview application. There are many instances of Gview that can run simultaneously so the user can fill it's screen with an abundance of Graph displays. The Gserver keeps track of all existing viewers and keeps track of the viewing range of each separate viewer called a 'viewport'. The Graph or 'Gview' instance is an important part of the system since it shows the beauty and efficiency of showing data fast and provide complete control. But a majority of the functions is not handled in the viewer application. It can be seen as a kind of 'thin client' application. It sends out request for what it wants to see, and receives a tailored package of information that exactly matches the request of the viewer. The requests are tailored to the window size, timespan and amount of TAGs the viewer is requesting. It does not provide more data than the user is able to distinguish, but certainly not lower than the display resolution is able to show to provide a crystal clear image of the data, retrieved at the highest possible speed.



Next pages will explain how it works, and what the main functionality is how such optized viewports can be achieved.

2.7.1 Gview basic concepts

The Gviewer is designed to cover current display resolutions, and even a few times more, so it tries to anticipate to future display resolutions to come. This is expected to have an end because at a certain moment the commercial available screen resolutions will bypass the maximum resolution a human can see with his/her eyes. Nowadays a minimum required resolution is full HD with 4K as very probable posibility and 8K displays expected to be affordable soon. This means that a top limit of 16K allows enough overkill (as we speak, being 2021) to support near future developments and make sure the application does never bump it's head in performance and capabilities.

2.7.1.1 Screen resolution

Vertical

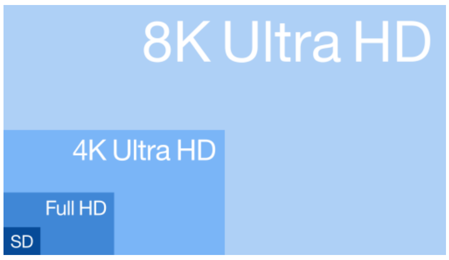
Now why is screen resolution so important for the operation of the viewer, and why only the horizontal resolution and not the vertical one? Reason for that is that a single datapoint is limited by the datapoint resolution i.e. 12bit, 16bit, 32bit or 64bit floating point number. Early in design the choice is made to use the standard LabVIEW [dbl] floating point format. It allows for numbers with the following limits:

Highest number:	1.79 $x10^{308}$
Lowest number:	4.94 x 10^{-324}

This 64bit [dbl] format has 15 decimal digits and therefore allows an abundant range and accuracy. The actual displayed line of samples is limited by the resolution of the input channel which cannot be defined on forehand due to the universal nature of application field. If the display vertical resolution is higher, the amount of detail is higher and vice versa without any influence on the data elements sent to the graph.

Horizontal

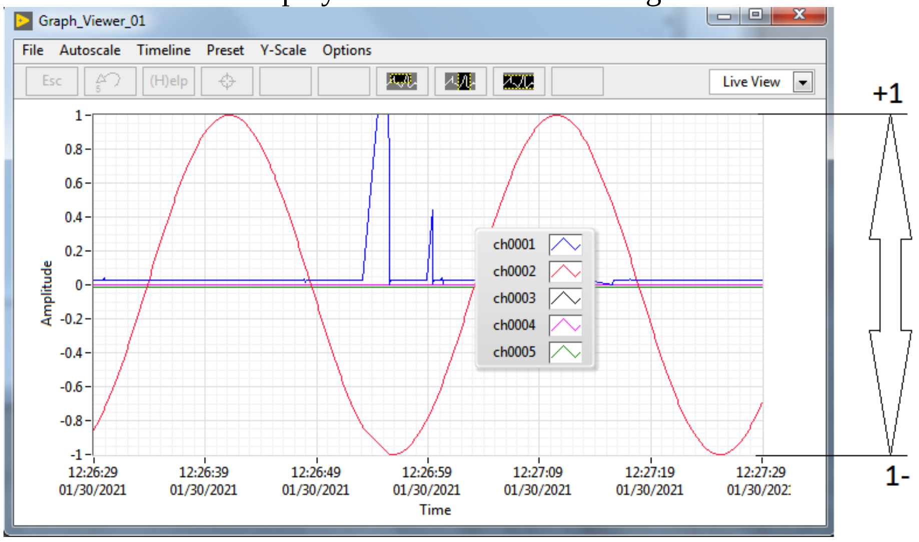
For the horizontal resolution however that is something completely different. One main property of the Graph viewer is that the horizontal line represents time. It's about trend lines shown on 'some selection' of progress in time. That time span can be 3 millisecond, or 3 years. If 3 years of data would be selected and data is fetched at millisecond resolution then for 5 shown TAGs that would mean trying to show 50 $x10^{12}$ datapoints where on an average HD screen only 1920 pixels exist, giving an overkill of 100 million samples per pixel. Then this abundant amount of data per pixel must be transported and rendered which will result in a very slow response and extreme load on processor power. Clearly that's not the way to go.



So the Gviewer does it much cleaner where at any change in data or window size, the Gviewer will send it's horizontal resolution to the Gserver. The Gserver does not have to bother for make calculations of a nice dataset. It will just select the dataset from one of the available resolution layers that the database already prepared in advance. The whole timespan range from milliseconds to years are coved according initial system settings. All possible timespan ranges including informative averaging and anti-aliasing calculations are already pre-calculated and can therefore be delivered in less than a split second. The data sent is so small that it will never show noticeable load on any data system (CPU performance). Flashing fast and crystal clear.

2.7.1.2 *Y axis display*

The Gview Y-axis display has a standard view range of -1 to +1



That -1 to +1 range does not change if individual or all vertical ranges of one of the shown traces are changed. The XY graph Offset & Gain setting however can change depending on which TAGs Y-axis is shown on the left side of the graph.

If TAGs are scaled by moving them up or down, Scrollwheel or windowed zooming then their individual offset and gain setting will be adjusted to fit according the -1 and +1 viewable range. But the viewer won't notice; since the graph Offset & Gain settings are adapted to the TAG that has ownership over the left Y-axis scale.
For the Y-axis scaling each tag has a max and min range value that represents that are converted to the -1/+1 displayed view.

2.7.1.3 *Menu navigation*

2.7.1.4 *Live update / Freeze*

2.7.1.5 *Dataset presets*

2.7.1.6 *Area of interest*

2.7.1.7 *Export data*

2.7.2 Gview initialization

2.7.3 Gserver Queries

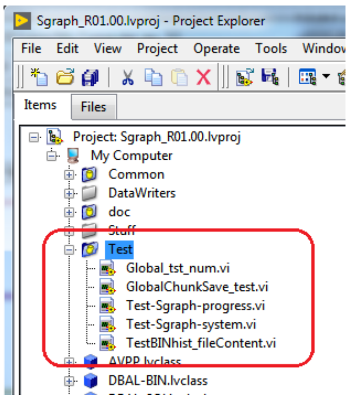
2.7.4 Gviewer screen updates

2.7.5 Gviewer dedicated settings

2.7.6 Gviewer general settings

3. Testing ∑graph modules

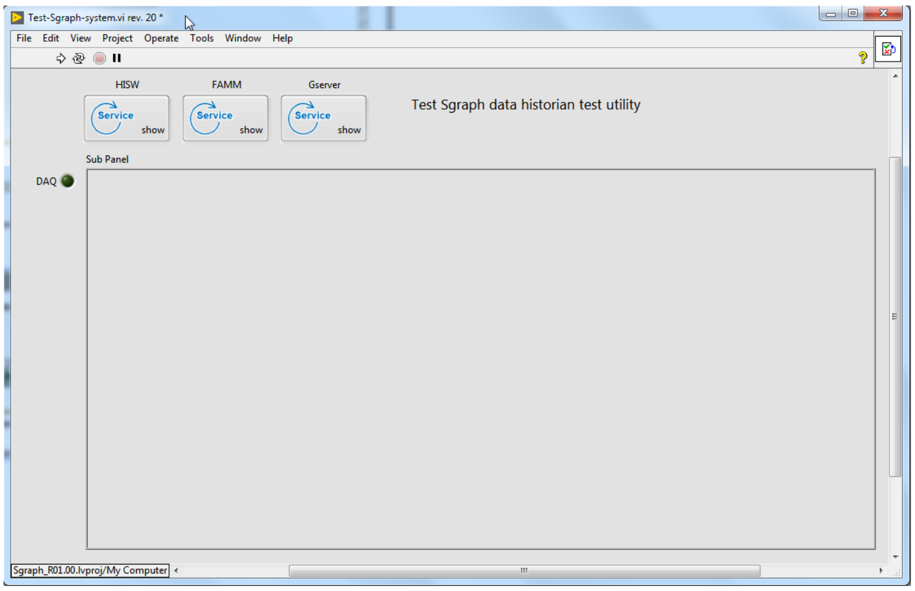
During development of the ∑graph components it is necessary to have an ability to test specific software components for both development and familiarization purposes. By running test applications one is able to investigate how the ∑graph system works. The several test components are described in this chapter and stored in the ***test*** section of the project:

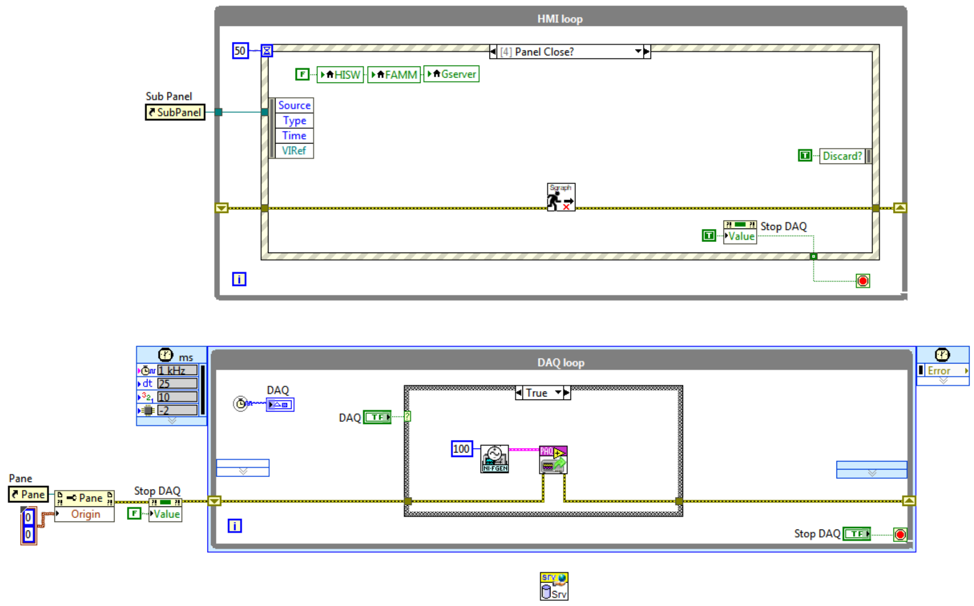


If test applications are created; this is the place to store them within the project.

3.1 Test-Sgraph-system.vi

The application ***Test-Sgraph-system.vi*** is the major test application for testing all sub-modules using simulated data generators.

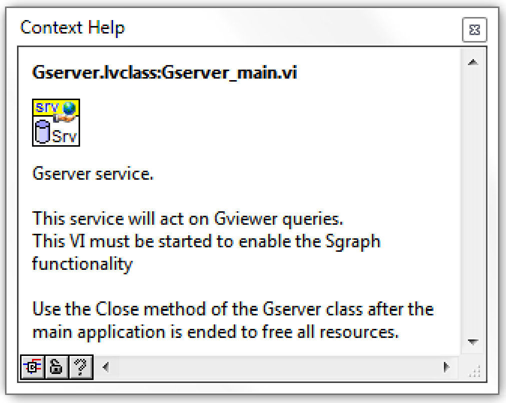

Test-Sgraph-system.vi rev. 20 *
File Edit View Project Operate Tools Window Help
HISW
FAMM
Gserver
Service show
Service show
Service show
Test Sgraph data historian test utility
Sub Panel
DAQ
Sgraph_R01.00.lvproj/My Computer



From here it is possible to run a ∑graph system, open Viewers and it serves as a template for a complete system setup.

3.1.1 Gserver startup

The Gserver main server should run somewhere in your main program where it can run continuously and maintain all services. The Gserver main vi is called:



Context Help

Gserver.lvclass:Gserver_main.vi

Gserver service.

This service will act on Gviewer queries.
This VI must be started to enable the Sgraph functionality

Use the Close method of the Gserver class after the main application is ended to free all resources.

This vi will startup all other necessary services as mentioned in this manual.

8. Abbreviations

Abbreviation	Explanation
DCS	Distributed Control System, by i.e. Siemens, Honeywell, Yokogawa, Emerson, Used in chemica land 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 of fast processing is required.
ATE	Automatic Test Equipment as part of Test & Measurement systems or product test & certification tooling.
WAQ	∑graph's definition of waveform acquisition method
PAQ	∑graph's definition of periodic acquisition method
EAQ	∑graph'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	Hierarchial Data Format verion 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 ∑graph viewer
Vsolver	module that determines which data set is needed by the ∑graph
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)