LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note **LIGO-T990013-A - D** 4/1/99

Diagnostics Test Software

Daniel Sigg and Peter Fritschel

Distribution of this draft:

GDS

This is an internal working note
of the LIGO Project.

LIGO Hanford Observatory
P.O. Box 1970 S9-02
Richland, WA 99352
Phone (509) 372-8106
FAX (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (504) 686-3100
FAX (504) 686-7189
E-mail: info@ligo.caltech.edu

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS NW17-161
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: http://www.ligo.caltech.edu/

Table of Contents

1 OVERVIEW 5

2 USER INTERFACE 8
2.1 INSTALLATION 8
2.2 GETTING STARTED 8
2.3 COMMAND LINE INTERFACE 9
2.3.1 Diagnostics Tests 9
2.3.2 Arbitrary Waveform Generator 11
2.3.3 Testpoint Control 13
2.4 GRAPHICAL USER INTERFACE 14
2.4.1 Common Properties of Diagnostics Tests 15
2.4.2 Diagnostics Tests 15
2.4.3 Control Screens 15

3 ANALYSIS ALGORITHMS 16
3.1 FFT MEASUREMENTS 16
3.1.1 Sampling rate reduction: multistage decimation 16
3.1.2 Zoom Analysis 19
3.1.3 Data Windowing 21
3.1.4 Power spectrum estimation using the FFT 21
3.1.5 Cross-spectral density 22
3.1.6 Transfer function estimates 23
3.1.7 Coherence estimates 23
3.2 SWEPT SINE MEASUREMENTS 23
3.2.1 Digital demodulation & frequency response calculation 23
3.2.2 Numerical integration algorithm 24
3.2.3 Integration time 26
3.2.4 Settling time 26
3.2.5 Coherence calculation 26
3.3 POLE-ZERO CURVE FITTING 26
3.4 CORRELATION MEASUREMENTS 26
3.5 TIME MEASUREMENTS 26
3.6 OCTAVE BAND ANALYSIS 26

4 TEST ORGANIZATION 27
4.1 INTERFACES 27
4.1.1 Storage API 27
4.1.2 Real-Time Data Distribution API 28
4.1.3 Excitation API 28
4.1.4 Test Point API 28

4.2 TEST SUPERVISORY 28
4.2.1 Standard Supervisory Task 28
4.3 TEST ITERATORS 30
4.3.1 Repeat 30
4.3.2 Parameter Scan 30
4.3.3 Optimization 30
4.4 TESTS 31
4.4.1 Sine Response 31
4.4.2 Swept Sine 32
4.4.3 FFT Tests 32
4.4.4 Time Series Measurements 34
4.4.5 Random Stimulus Response Test 34

5 EXCITATION ENGINE 35
5.1 OUTPUT WAVEFORMS 35
5.1.1 Overview 35
5.1.2 Periodic Waveforms 35
5.1.3 Non-Periodic Waveforms 37

APPENDIX A MESSAGE PASSING INTERFACE 38
A.1 USING REMOTE PROCEDURE CALLS 38
A.2 USING SOCKETS 38

APPENDIX B DATA REPRESENTATION 40
B.1 COMMON 42
B.1.1 Globals 42
B.1.2 Synchronization Tools 43
B.1.3 Environment 43
B.1.4 Parameter Scan 44
B.1.5 Parameter Optimization 45
B.2 RESULTS 46
B.2.1 Plot 46
B.2.2 Time Series 50
B.2.3 FFT and (Cross) Power Spectrum 50
B.2.4 Transfer Function and Coherence 51
B.2.5 List of Coefficients 52
B.2.6 Measurement Values 53
B.3 DIAGNOSTICS TESTS 53
B.3.1 Sine Response, Harmonic Distortion and Two-Tone Intermodulation Tests . 53
B.3.2 Swept Sine Tests 54
B.3.3 Fourier Tests 55
B.3.4 Time Series Measurements and Trigger Response Tests 56
B.3.5 Random Stimulus Response Tests 56
B.4 XML CONVENTIONS 56

APPENDIX C SOFTWARE MODULES 57
C.1 OVERALL STRUCTURE 57
C.2 UTILITIES 59
C.2.1 gdserr 59
C.2.2 gdsheartbeat 59
C.2.3 gdsmutex 59
C.2.4 gdsprm 59
C.2.5 gdsstring 60
C.2.6 gdstask 60
C.2.7 rpcinc 60
C.2.8 tconv 61
C.3 ALGORITHMS 61
C.3.1 decimate 61
C.3.2 gdsrand 61
C.3.3 gdssigproc 61
C.3.4 sineanalyze 61
C.4 ARBITRARY WAVEFORM GENERATOR 62
C.4.1 awg 62
C.4.2 awgapi 62
C.4.3 awgfunc 63
C.4.4 awg_server 63
C.4.5 excitation 63
C.5 COMMAND LINE INTERFACE 63
C.5.1 cmdline 63
C.5.2 gdscmd 64
C.5.3 gdsmsg 64
C.5.4 gdsmsg_server 64
C.5.5 gdsmsg_sockets 64
C.6 DATA ACQUISITION INTERFACE 64
C.6.1 gdschannel 64
C.6.2 gdsrtdd 65
C.7 DIAGNOSTICS TESTS 65
C.8 HARDWARE DRIVERS 65
C.8.1 cobox 65
C.8.2 ds340 65
C.8.3 gdsdac 66
C.8.4 gpsclk 66
C.8.5 rmapi 66
C.9 REFLECTIVE MEMORY ORGANIZATION 67
C.9.1 testpoint 67
C.9.2 testpoint_server 67

C.10 SCHEDULER 67
C.10.1 gdssched 67
C.10.2 gdssched_client 68
C.10.3 gdssched_server 68
C.11 STORAGE OBJECTS 68
C.11.1 gdsdatum 68
C.11.2 diagdatum. 70
C.11.3 rtddinput 70
C.12 PROGRAMS. 71
C.12.1 chndump. 71
C.12.2 diag. 71
C.12.3 gdsd 71
C.12.4 libgds.so 71

1 OVERVIEW

The LIGO global diagnostics system (GDS) provides diagnostics test capability for performing stimulus-response tests. Diagnostics tests are divided into five groups: *(i)* sine response tests which include multiple stimulus and multiple response, harmonic distortion and two-tone intermodulation, *(ii)* swept sine response which determines transfer functions, *(iii)* FFT tools which perform power spectrum estimates and cross-correlation measurements, *(iv)* time series measurements which measure the response to a trigger signal, and *(v)* pseudo-random stimulus response tests which utilize wide-bandwidth excitation signals.

The diagnostics test tools provided by GDS are not meant to replace more traditional means to diagnose the instrument—such as stand-alone network and spectrum analyzers—but rather to complement these tools for cases where test points are not readily available or where measuring points are at far distant locations. In particular, digital servo controllers are implementing digital test points which can be used to inject excitation signals and to extract intermediate feedback signals. The diagnostics test system uses the LIGO data acquisition system to collect signals simultaneously from different subsystems. It implements an excitation engine for generating test signals which are synchronized with GPS time and which are provided to both digital and analog subsystems at all major locations.

Diagnostics tests are run from the control room or from any machine which is connected to the control and monitor network. Data from the instrument are obtained thought the network data server which in turn gets the data through a reflective memory network which connects to the data collection units (see Fig. 1 and T980026-00). Data collection units can acquire data from analog signals (ADCU), from digital subsystems (DDCU), or from EPICS channels (NDCU).

Test signals are generated by an excitation engine which is connected to the same reflective memory network which provides both read-back and interface to the digital servo systems. The excitation engine also implements digital-to-analog converters which provide test signals to analog subsystems. Excitation engines are available in every building. Additionally, remote-controlled stand-alone signal generators can be used in temporary setups. Digital servo systems implement a test point interface which allows the user to select a finite set of test inputs and test outputs which are then read or written to and from reflective memory, respectively (see also T980020-A).

Two separate user interfaces to the diagnostics test system are provided: *(i)* a command line interface which allows the user to manually adjust test parameters, start a test and save the results; and *(ii)* a graphical user interface with identical capabilities. The results of a diagnostics test can be stored in the LIGO lightweight data format which follows the extensible markup language (XML) specifications.

A schematic overview of the diagnostics test software is shown in Fig. 2. It consists of the following components:

1. a test organizer which runs tests by dividing them into individual measurement steps and schedules them with the excitation engine, the network data server and the analysis process,
2. a command line interface to the test organizer,

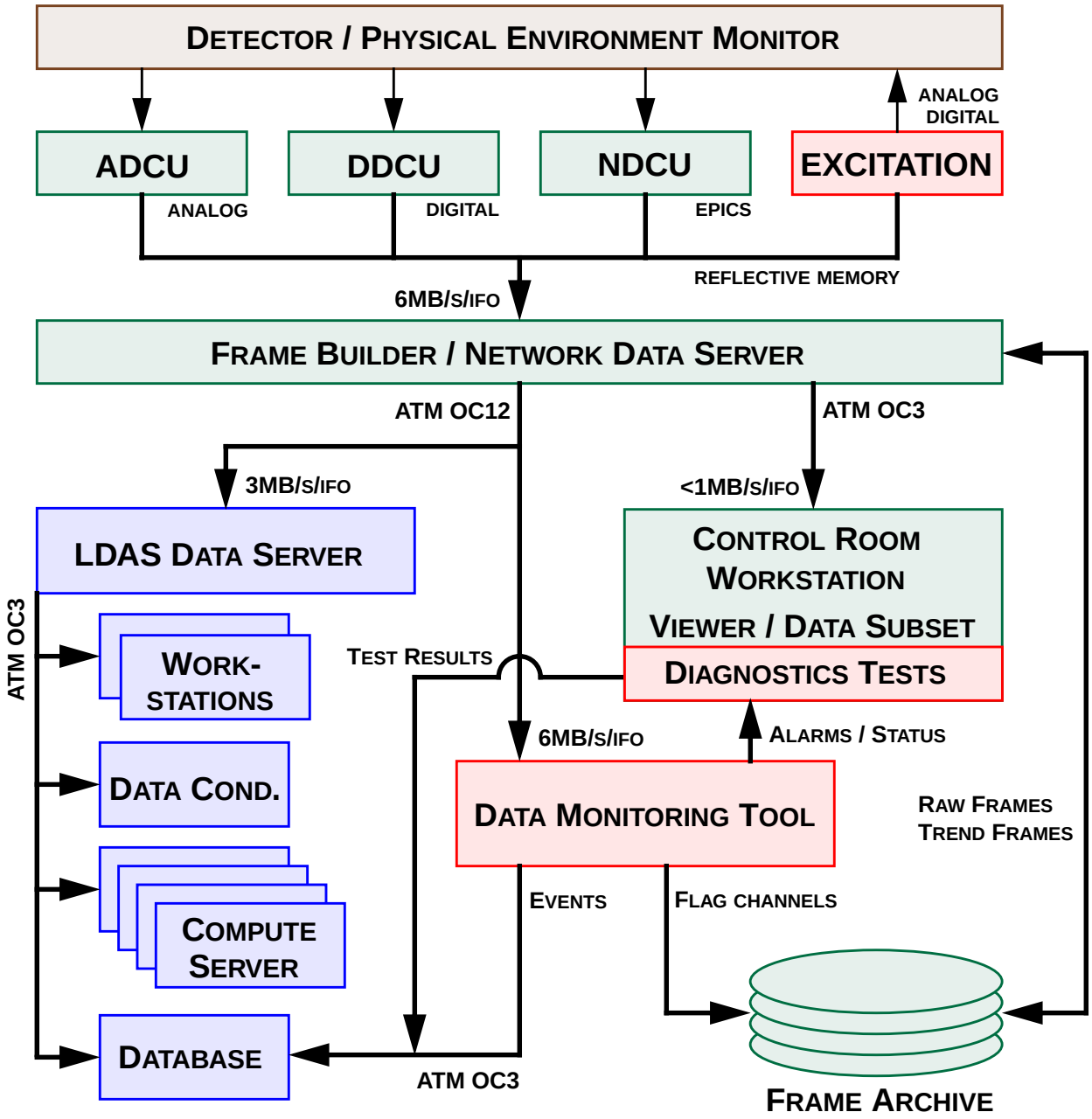


Figure 1: Overview of the data acquisition system (green boxes), the diagnostics system (red) and the data analysis system (blue). The physical environment monitor and the detector are show on top.

3. a graphical user interface for the most commonly used tests,
4. a graphical user interface to the excitation engine for manual operation,
5. a graphical user interface to the testpoint manager,
6. an excitation engine which controls stand-alone function generators (DS340s) and which can generate arbitrary waveforms and write them to a digital-to-analog converter or to digital test points residing in reflective memory,
7. an interface to the network data server to collect data,
8. an algorithm library which is used to analyze the measured data and produce the diagnostics test results,
9. a data wrapper which manages data and parameters associated with a test, and
10. an XML interface which saves and restores diagnostics parameters, data and results to and from disk in the LIGO lightweight data format, respectively.

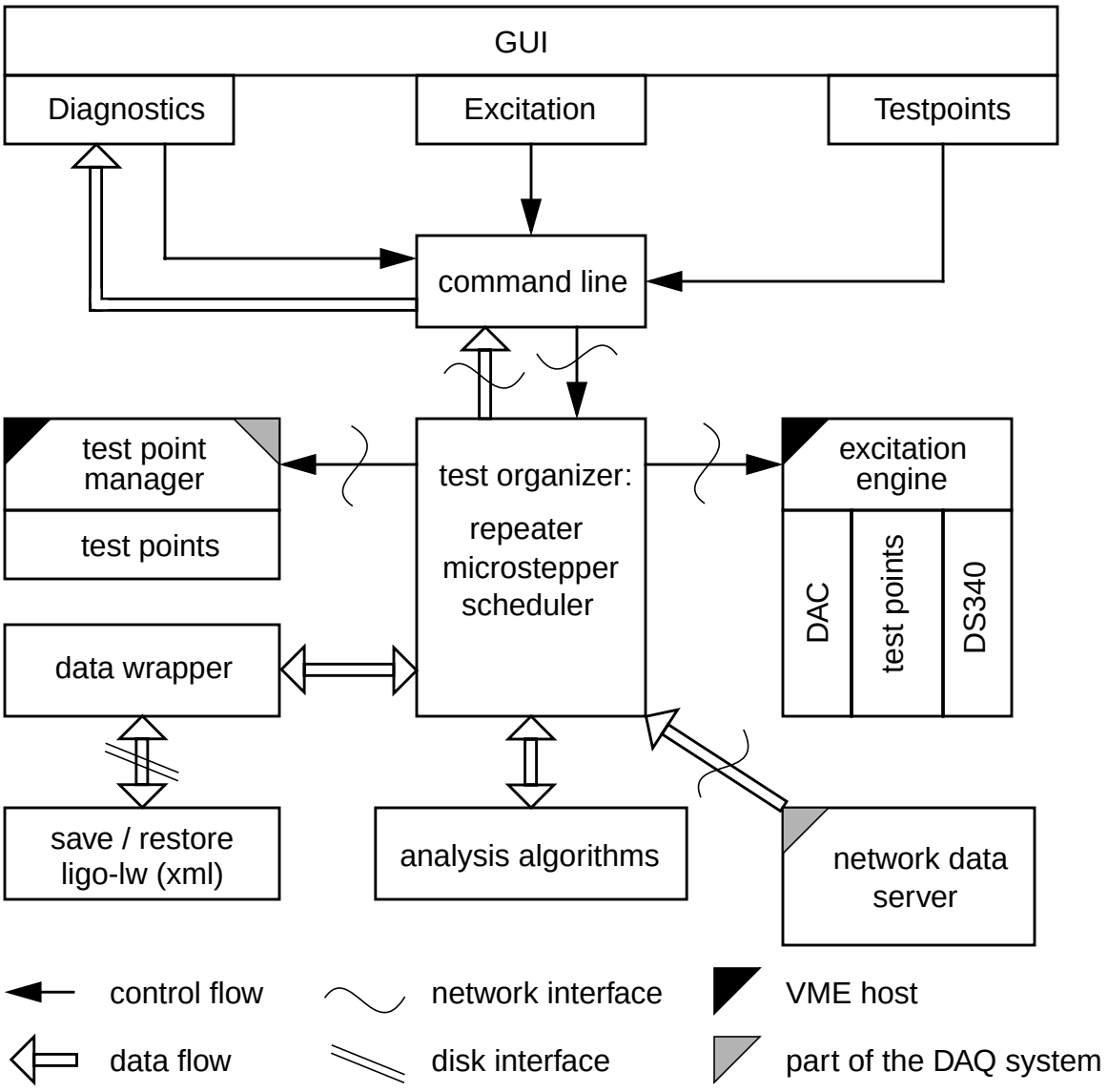


Figure 2: Overview of the diagnostics test software organization.

2 USER INTERFACE

2.1 INSTALLATION

TBD.

2.2 GETTING STARTED

The diagnostics program is started by typing ‘diag –opt’, where ‘–opt’ is a list of options specifying initial start-up parameters. The available options are:

Option	Description
–help	shows a help text.
–g	starts the graphical user interface allowing the user to select a diagnostics test.
–g ‘test’	starts the graphical user interface and preloads the specified test or interface. Possible values for ‘test’ are: awg: arbitrary waveform generator fft: Fourier tests, includes power spectrum and cross-correlation randomresponse: pseudorandom response test, pseudorandom cross spectrum test sineresponse: (multiple) sine response test, harmonic distortion test, two-tone intermodulation test sweptsine: swept sine test timeseries: time series measurement testpoint: testpoint control triggerresponse: single and multiple trigger response test
–c	starts the command line interface of a diagnostics test.
–l	runs the diagnostics kernel on a the local machine (default).
–s ‘server’	runs the diagnostics kernel on the specified remote machine. Requires that the diagnostics kernel daemon is installed on the remote machine.
–r ‘filename’	reads the specified file as a command line script for initialization. All commands which are available through the command line interface can be used in a script file. Lines starting with a ‘#’ are comments. This script is executed after all other options are parsed, but before the first user input.
–t ‘filename’	uses the specified file as a template for a new diagnostics test.
‘filename’	if exists, opens the specified diagnostics test file; if not, uses it as the default filename for saving the diagnostics test results.

The diagnostics kernel can either run on a local machine or on a remote machine. In the first case it is loaded dynamically and requires the shared object library ‘libgds.so.1’ to be available either in the local directory or the default system library directory.

2.3 COMMAND LINE INTERFACE

The command line interface presents a common way to control diagnostics tests, the arbitrary waveform generator and the test point interface. All these functions are provided by connecting the user interface to a diagnostics kernel, and by passing messages forth and back. The following commands are implemented as part of the user interface:

Command	Description
help	shows help text.
open	loads a diagnostics kernel on the local machine and established a connection. It returns a list of the supported capabilities. Return: supported capabilities: testing testpoints awg Possible error messages: – diagnostics kernel already connected – unable to connect to local diagnostics kernel
open ‘server’	opens a new connection to a diagnostics kernel on the specified remote machine. Return: Possible error messages: – diagnostics kernel already connected – unable to connect to remote diagnostics kernel
close	closes the connection to the diagnostics kernel Return: none Possible error messages: – not connected to a diagnostics kernel
read ‘filename’	opens the specified file and interprets each line of the file a command. Lines which start with ‘#’ are ignored. Return: echo of commands found in file Possible error messages: – file not found
exit/quit	quits the command line interface after closing any open connections.

Any command which is not recognized by the command line interface will be send to the diagnostics kernel. In case of an error the message “unable to send command to diagnostics kernel” is returned, otherwise the reply string of the command is displayed.

2.3.1 Diagnostics Tests

Diagnostics tests are controlled through named variables. A named variable can contain almost anything, for instance, it can contain a test parameter, a time series of the raw data, or a result array. Thus, only a small set of commands is necessary to setup and run a diagnostics test; they can be separated into the following categories:

i) commands to set and read parameters,
ii) commands to start and stop a diagnostics test,
iii) commands to save or restore settings, data and results of a diagnostics test, and
iv) notification messages which are passed back from the diagnostics kernel to inform the caller that a test has finished, or that a new result is available.

When running the diagnostics kernel on a remote machine the save and restore functions could in principle access either the local or the remote file system. The command line interface supports both possibilities, but defaults to the local machine.

The following list of commands is used to control diagnostics tests:

Command	Description
help ‘test’	shows help text for specified test. Use “help *” for a list of available tests.
run	starts a diagnostics test. Return: running Possible error messages: – –
abort	aborts a diagnostics test. Return: aborted Possible error messages: – test is not running
pause	Pauses a diagnostics test. Return: paused Possible error messages: – test is not running
continue	Continues a paused diagnostics test. Return: resumed – test is not running
set ‘variable name’ = ‘value’	Sets the specified variable to the given value(s). Return: none Possible error messages: – illegal argument
get ‘variable name’	Gets the value(s) of the specified variable(s). The variable name can contain the wildcard character ‘*’. Return: variable(s) and value(s) Possible error messages: – illegal argument
del ‘object name’	Erases a data objects belonging to scans, environment, plot, channel data and result. Return: ‘object’ erased Possible error messages: – illegal argument
defined ‘variable name’	Tests if a variable of the given name is defined. Use “defined *” to print a list of all defined variables. Return: yes/no, or list of names Possible error messages: none
brief	if on (default), the value returned by get is truncated if it is very long. Return: brief is on/off. Possible error messages: none

Command	Description
save –‘flags’ ‘filename’	saves the data of a diagnostics test. The flag specifies the data to be saved: all (all data associated with a diagnostics test), ext (extended data, all but images), std (standard data: parameters, settings and result), or prm (parameter data only). If the flag is omitted the default behavior is standard. If the filename starts with a ‘:’ and if the command line interface is connected to a remote machine, the test is saved on the remote machine. Return: ‘filename’ saved Possible error messages: – illegal filename
restore –‘flags’ ‘filename’	restores the data of a diagnostics test. The flag specifies the data to be restored: all (all data associated with a diagnostics test), ext (extended data, all but images), std (standard data: parameters, settings and result), or prm (parameter data only). If the flag is omitted the default behavior is standard. If the filename starts with a ‘:’ and if the command line interface is connected to a remote machine, the test is restored from the remote machine. Return: ‘filename’ restored Possible error messages: – file not found

Each diagnostics test defines a set of valid named variables—including their data type—which are necessary to run the test, or which are returned as a result. Variable names should not contain spaces, tabs, nor any other special character. Variable names are case-insensitive.

2.3.2 Arbitrary Waveform Generator

Arbitrary waveform generators are implemented separately for every interferometer node. An interferometer node can have multiple excitation engines (CPUs or stand-alone signal generators), each of them consisting of multiple slots which can be used independently to send waveforms to different channels. Commands to the arbitrary waveform generator are of all the form ‘awg command arguments’. The following commands are supported:

Command	Description
help	shows help text.
show ‘node’.’awg’	shows the configuration of an arbitrary waveform generator. Return: AWG information Possible error messages: – arguments for show are 'node'.'awg' – node%i/awg %i not available
new ‘channel’	reserves a slot in an arbitrary waveform generator for the specified channel. Requires the full channel name and returns a non-negative slot number if successful. (The returned slot number also encodes ‘node’ and ‘awg’ number and is unique for each system.) Return: slot %i Possible error messages: – no slot available or invalid channel name

Command	Description
free ‘slot’	frees a previously reserved slot of an arbitrary waveform generator Return: slot %i freed Possible error messages: – invalid slot number – slot not available or invalid
set ‘slot’ ‘waveform’	selects a new waveform in the given slot of an arbitrary waveform generator. If the waveform argument is omitted the slot is cleared and the output reset to zero. Return: slot %i enabled Possible error messages: – invalid slot number – invalid arguments – not enough memory – unable to download waveform – unrecognized waveform – slot not available or invalid
clear ‘node’.’awg’	clears all waveforms from a given arbitrary waveform generator and frees all slots. Return: reset succeeded Possible error messages: – reset failed – node %i/awg %i not available
stat ‘node’.’awg’	shows statistical performance data of a given arbitrary waveform generator. Return: statistics information Possible error messages: – arguments for stat are 'node'.'awg' – node %i/awg %i not available – no statistics available

The ‘%’ arguments in the return message are replaced by the corresponding value and follow the C printf convention. Error message are preceded by the word “error:”. If the command is none of the above, the message “unrecognized command” will be returned.

The command arguments are explained below:

Argument	Description
‘node’	interferometer node, e.g. 0 for H1 and L1, and 1 for H2.
‘awg’	identification number of excitation engine, or stand-alone signal generator: starts with 0, and the stand-alone units generally use 5.
‘channel’	full channel name of an excitation or read-back channel, e.g. “H1:LSC–TEST_1”.
‘slot’	slot number as returned by new.

Argument	Description
‘waveform1’	specifies a periodic waveform of format: ‘func’ ‘frequency’ ‘amplitude’ ‘offset’ ‘phase’ where the frequency is given in Hz, the amplitude and phase in V, and the phase in rad.
‘waveform2’	specifies a band-limited noise source of format: ‘noise’ ‘start frequency’ ‘stop frequency’ ‘amplitude’ ‘offset’
‘waveform3’	specifies a swept sine wave of format: sweep ‘start frequency’ ‘stop frequency’ ‘start amplitude’ ‘stop amplitude’ ‘sweep time’ ‘sweeptype’ ‘updn’ where the sweep time is given in sec.
‘waveform4’	specifies an arbitrary waveform of format: arb ‘sample frequency’ ‘scaling’ ‘trigger type’ ‘value 1’ ‘value2’... ‘value n’ where the values are given in V. The output voltage is calculated by multiplying the specified values with the scaling factor. The trigger can either be continuous which repeats the waveform indefinitely, wait which waits for a trigger, or trigger which outputs the (previously stored) waveform exactly once.
‘func’	one of the following: sine, square, ramp, or triangle.
‘noise’	one of the following: normal (normal distributed noise), or uniform (uniformly distributed noise).
‘sweeptype’	one of the following: linear, or log.
‘updn’	one of the following: + (up), – (down), or ‘blank’ (bidirectional)
‘trigger type’	one of the following: c (continuous), w (wait), or t (trigger)

2.3.3 Testpoint Control

Each interferometer node has its own testpoint control interface. Commands to the testpoint interface are of all the form ‘tp command arguments’. The following commands are supported:

Command	Description
help	shows help text.
show ‘node’	shows the configuration of a test point interface. Return: testpoint information Possible error messages: – invalid node number supported wildcard: “show *” to show test point information of all nodes.

Command	Description
set ‘node’ ‘number1’... ‘number n’	sets testpoints of the specified node. Return: test point set Possible error messages: – invalid node number – unable to set test point
clear ‘node’ ‘number1’... ‘number n’	Clears testpoints from the specified node. Return: test point cleared Possible error messages: – invalid node number – unable to clear test point supported wildcards: “clear *” to clear all testpoints from all nodes, and “clear ‘node’ *” to clear all testpoints from the specified node.

Error message are preceded by the word “error:”. If the command is not recognized, the error message “unrecognized command” will be returned.

2.4 GRAPHICAL USER INTERFACE

The graphical user interface is using the same message passing interface to communicate with the diagnostics kernel. Thus, all functions of the diagnostics kernel which are accessible through the command line interface can also be used by the graphical user interface. A description of the message passing mechanism can be found in Appendix A.

2.4.1 Common Properties of Diagnostics Tests

2.4.1.1 File Handling

2.4.1.2 Synchronization Tools

2.4.1.3 Channel Selection

2.4.1.4 Exporting Data

2.4.1.5 Parameter Sweep

2.4.1.6 Parameter Optimization

2.4.2 Diagnostics Tests

2.4.2.1 Sine Response, Harmonic Distortion and Two-Tone Intermodulation Tests

2.4.2.2 Swept Sine Tests

2.4.2.3 Fourier Tests

2.4.2.4 Time Series Measurements

2.4.2.5 Trigger Response Tests

2.4.2.6 Random Stimulus Response Tests

2.4.3 Control Screens

2.4.3.1 Arbitrary Waveform Generator

2.4.3.2 Testpoint Control

3 Analysis Algorithms

3.1 FFT Measurements

An FFT program (such as the FFTW, Fastest Fourier Transform in the West) will be used to compute the N-point DFT (discrete-time fourier transform) of a data stream x:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi i k n / N} \tag{1}$$

3.1.1 Sampling rate reduction: multistage decimation

Performing an N-point FFT analysis of a data channel at the sampled rate gives a full span analysis, with spectral information in the band 0–$f_s/2$. For baseband measurements (frequency spans which start at DC) with increased resolution, the duration of the time record must be increased. This is done by decimating the data appropriately, while keeping the FFT length fixed at N. To avoid aliasing in the decimation process, the data must be low-pass filtered with a digital filter. For large decimation factors, it is more efficient to implement multiple stages of decimation, as shown in Fig. 3.

To simplify (though not necessarily optimize) the computation we fix the single stage decimation at a factor of $D_i = 2$. Thus decimation factors of 2^n (n an integer) are possible by computing n

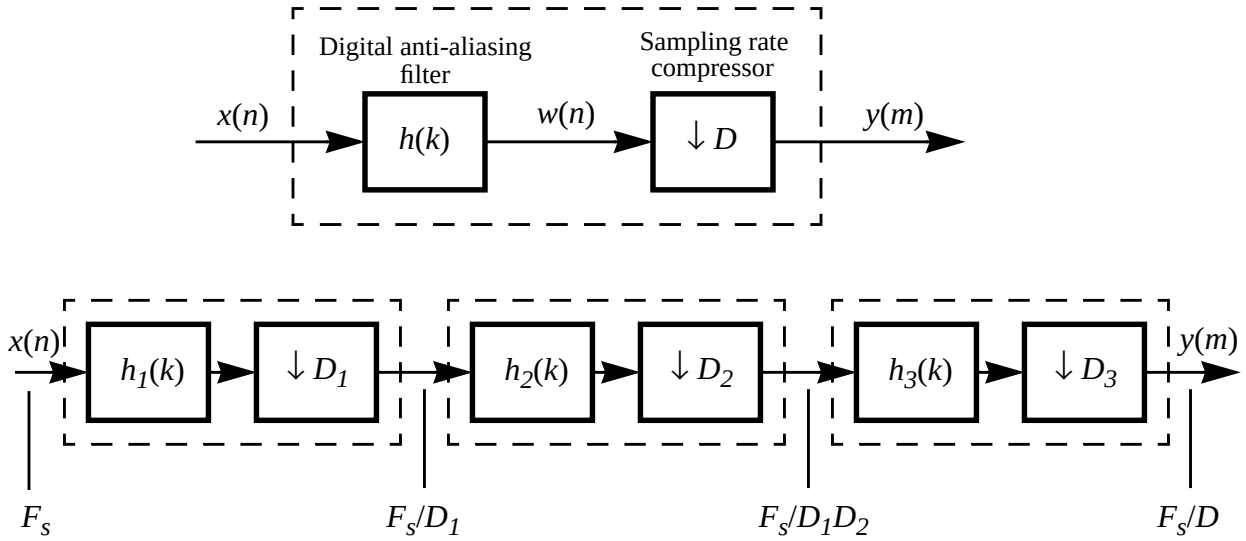


Figure 3: Top: Block diagram of a decimator. The digital low-pass filter is an FIR filter:

$$w(n) = \sum_{k=0}^{N-1} h(k) x(n-k),$$

where N is the number of filter coefficients. The sampling rate compressor simply selects every Dth output sample. Bottom: Multistage decimation process.

stages of decimation, leading to FFT spans of DC – $F_s/2^{n+1}$ (where F_s is the channel sampling rate). This simplification means that the coefficients of the digital low-pass filters in each stage are identical: $h_1 = h_2 = \ldots = h_n$.

Sampling rate converters generally implement FIR digital filters because of their linear phase response, a necessary feature in our application as well. The diagrams in Fig. 3 imply a direct-form realization of the FIR filter. This is an inefficient method of calculation, since the filter is operating at the full input sample rate, but only every *D*th output sample is kept. It is more efficient to embed the downsampling operation within the filter, as shown by the realization of Fig. 5. This structure also takes advantage of the symmetry of the filter coefficients for a linear-phase FIR filter – i.e., that $h(n) = h(N-1-n)$; this allows halving the number of multiplications to compute the filter.

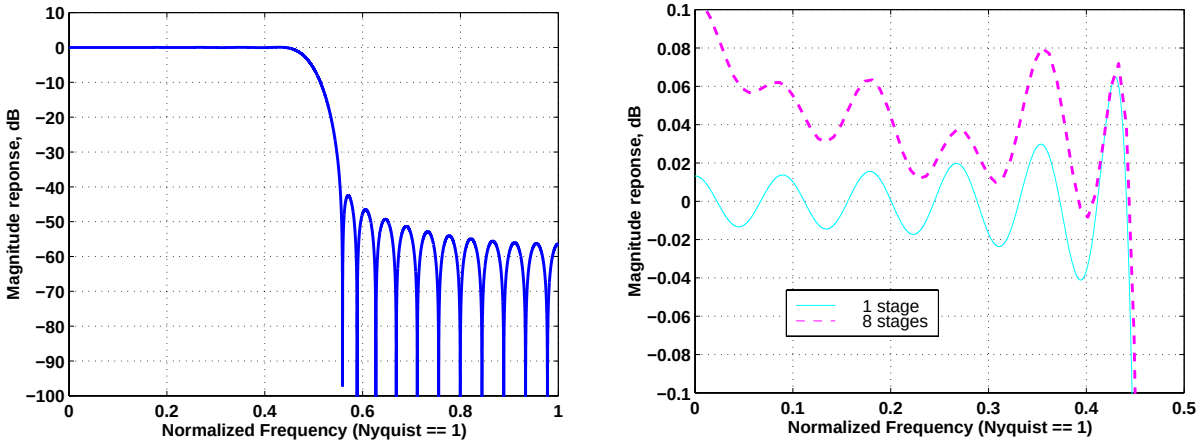
Another technique to reduce the computation time in a decimate-by-2 stage is to use a *half-band* filter. A half-band filter is one which satisfies two constraints: the ripple in the pass- and stop-bands are equal; the pass- and stop-band cutoff frequencies are related to the Nyquist frequency as: $f_p + f_s = f_N$. These symmetries result in about 50% of the filter coefficients being zero, cutting down the number of multiplies by roughly a factor of 2. Care must be taken in the design of the digital filter so that when multiple stages are used (up to ~10 stages may be used in practice), the passband ripple from each stage does not accumulate into a much larger ripple at the output. Two types of filter design methods have been used to generate specific filters:

1. An FIR filter designed using a least squares technique trades off increased error at the band edge for better response over most of the passband; thus a cascade of these filters retains small passband error, since the band edge error of one filter is cut off by the next filter.
2. The McClellan-Parks filter design algorithm gives a constant error over the pass- and stop-bands. Compared to the least squares design, this filter has better performance (lower ripple) near the band edge; however, multiple stages build up more error in the lower part of the pass-band.

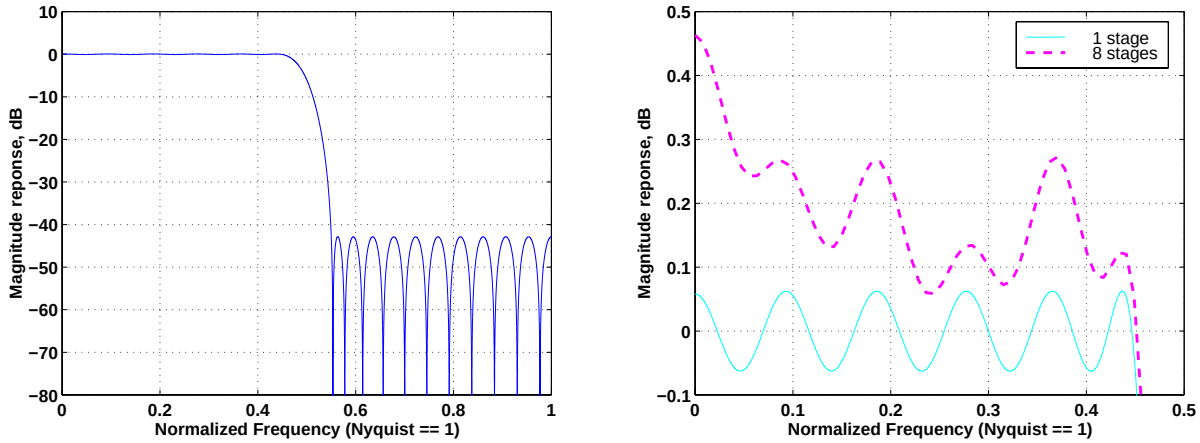
Frequency responses for both types of filter, each of order 42, are shown in Fig. 4; each is computed with 12 multiplications per output point. Several FIR filters are available to choose from (more can always be added) to trade-off between execution speed and error. Current selections are:

Type	Order	Passband cutoff frequency[1]	Passband Ripple (dB)		Stopband attenuation (dB)
			Midband	Edge	
Least-squares	22	0.45	0.1	0.8	30-40
Least-squares	42	0.45	0.02	0.1	40 - 56
Least-squares	82	0.45	0.0006	0.01	60 - 90
McClellan-Parks (equiripple)	42	0.45	0.05		43

1. as a fraction of the Nyquist frequency



(a)



(b)

Figure 4: Magnitude response of two half-band, order 42 FIR filters. On the left is the response of a single filter stage, and the right shows the passband response for a single stage (solid line) and for 7 cascaded decimation stages (dashed line). (a): filter designed using a least-squares method; (b) filter designed using the McClellan-Parks method. The normalized pass- and stop-band cut-off frequencies are 0.45 and 0.55, respectively.

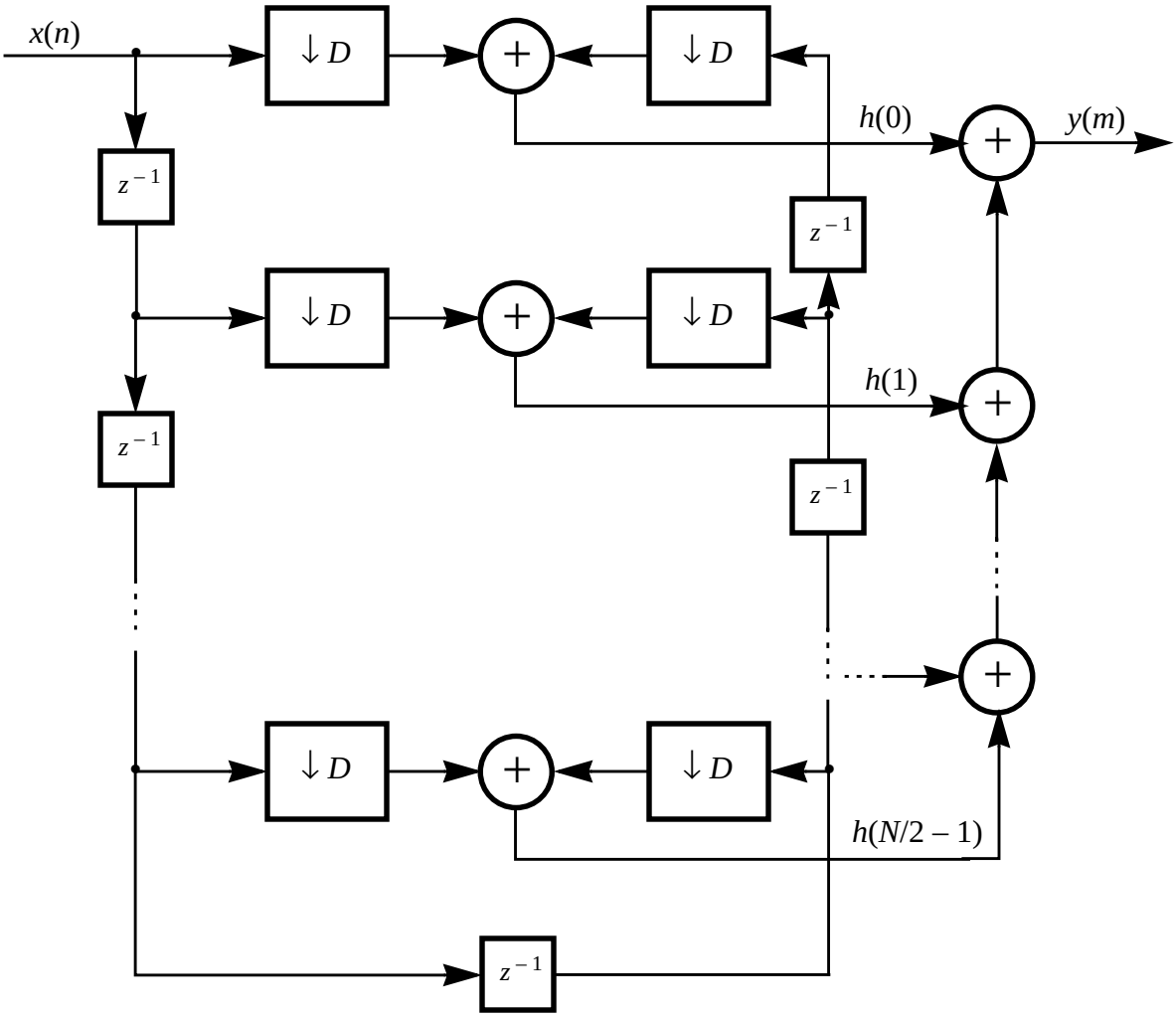


Figure 5: Efficient structure for computation of a decimator that exploits the symmetry of the FIR filter coefficients. For a half-band filter, half of the coefficients *h* would be zero, and thus not computed.

3.1.2 Zoom Analysis

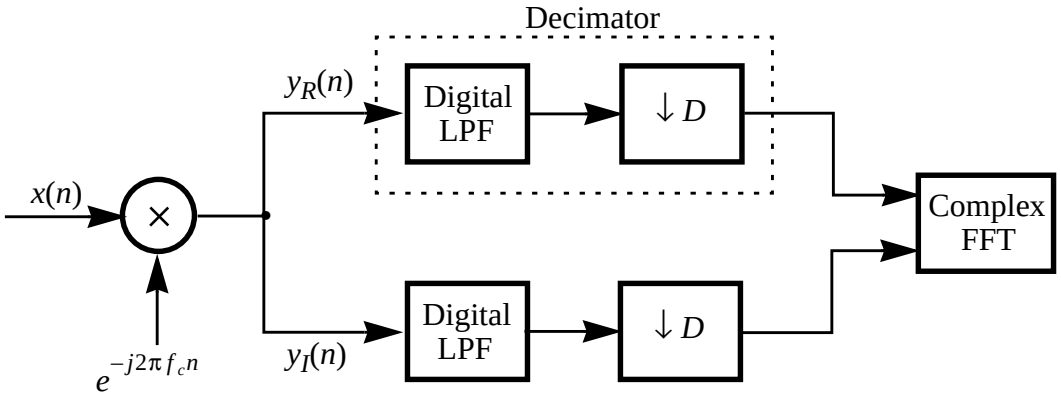
In order to start an FFT span above DC, the data must first be down-converted so that the center frequency of the span is shifted to DC. This is accomplished with the heterodyne procedure shown in Fig. 6. The heterodyne multiplication produces a complex sequence *y* from the real data sequence *x*:

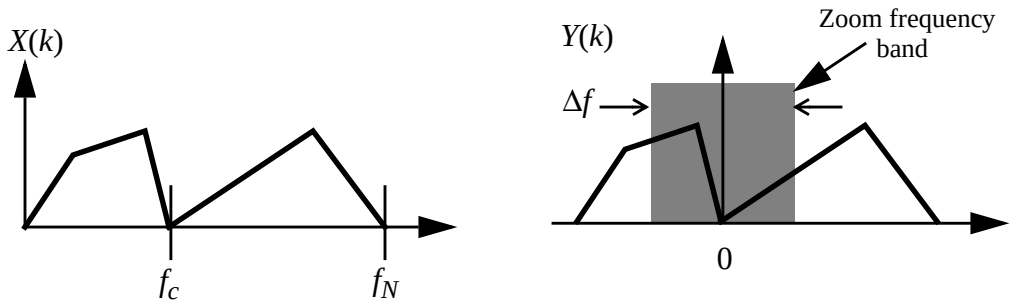
$$y(n) = e^{-j2\pi f_c n} \cdot x(n) \qquad (2)$$

The DFT of this sequence,

$$Y(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi i \frac{n}{N}(k + f_c)} \tag{3}$$

shows that the frequencies k of the original sequence x have been shifted to the frequencies $k - f_c$ of the sequence y. After heterodyning, we need to low-pass filter and decimate the data to select the frequency span of interest. The data are now complex, so a complex FFT is required; the negative frequencies of Y in this case are independent of the positive frequencies, and in fact they contain the lower half of the span, as shown in Fig. 6.





FFT coefficients of Y, and relation to frequencies of original data x:

$$\underbrace{Y_0, Y_1, Y_2, \ldots\ldots Y_{\frac{N}{2}-1}}_{f_c \le f < f_c + \Delta f/2}, \underbrace{Y_{\frac{N}{2}}, Y_{\frac{N}{2}+1}, \ldots\ldots Y_{N-1}}_{f_c - \Delta f/2 < f < f_c}$$

Figure 6: Block diagram of heterodyning for zoom analysis. The center frequency f_c is downconverted to DC, and the lower half of the frequency span is mapped to negative frequencies. The full span of the zoom, Δf, is again restricted to $(1/2^p)$ multiples (p an integer) of the original channel sampling frequency. The decimator consists of $(F_s/2\Delta f)$ stages of the single

3.1.3 Data Windowing

All FFTs are performed on windowed data. The following windowing functions are offered:

Window Type	Time-domain sequence $0 \le i \le N-1$	Max. Amplitude error	Comments
Uniform	no window	4 dB	useful for looking at transients
Hanning	$w_i = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi i}{N-1}\right)\right]$	1.5 dB	general purpose; lowest noise floor
Flat-top	$w_i = \frac{1}{2}\left(1 - 1.93\cos\left(\frac{2\pi i}{N-1}\right) + 1.29\cos\left(\frac{4\pi i}{N-1}\right) - 0.388\cos\left(\frac{6\pi i}{N-1}\right) + 0.028\cos\left(\frac{8\pi i}{N-1}\right)\right)$	0.02 dB	very wide pass band, but most accurate amplitude meas.
BMH	$w_i = \frac{1}{2}\left(1 - 1.36109\cos\left(\frac{2\pi i}{N-1}\right) + 0.39381\cos\left(\frac{4\pi i}{N-1}\right) - 0.032557\cos\left(\frac{6\pi i}{N-1}\right)\right)$	0.8 dB	large dynamic range (good for separating two close frequencies, with widely different amplitudes)

Table 1: Windowing functions that can be applied to the data prior to Fourier transforming. Amplitude error in the transformed data occurs for frequencies not exactly at a bin frequency.

3.1.4 Power spectrum estimation using the FFT

Welch's method of power spectrum estimation is implemented according to the following algorithm:

1. the data is broken up into D, possibly overlapping segments (50% overlap typical), of N points each; each segment may then optionally be 'de-trended' (i.e., the mean, or best linear fit may be removed) before being windowed
2. each (windowed) segment is transformed with an N-point FFT, giving $X_i(k)$ ($i = 1 - D$).

3. the magnitude squared of each segment is properly scaled, forming the periodogram $P_i(k)$ of each segment:

$$P_i(0) = \frac{1}{W}\left|X_i(0)\right|^2$$

$$P_i(k) = \frac{1}{W}\left[\left|X_i(k)\right|^2 + \left|X_i(N-k)\right|^2\right] \quad k = 1, 2, \ldots, \left(\frac{N}{2} - 1\right) \tag{4}$$

$$P_i(N/2) = \frac{1}{W}\left|X_i(N/2)\right|^2$$

where W accounts for the power in the window:

$$W \equiv N \sum_{n=0}^{N-1} w(n) \tag{5}$$

This normalization is chosen so that the value at each point in the periodogram gives the energy density in units of mean-squared-amplitude.

4. the average of the D periodograms is computed to form the power spectrum of x:

$$\langle P_{xx}(k) \rangle = \frac{1}{D} \sum_{i=1}^{D} P_i(k) \tag{6}$$

5. scaling factors may be applied to convert from mean-squared-amplitude (msa) to msa/Hz, (root-mean-square)/$\text{Hz}^{-1/2}$, peak, engineering units, etc.

If the data has been mixed down for zoom analysis, the periodogram is formed somewhat differently, as indicated in Fig. 6.

3.1.5 Cross-spectral density

The cross-spectral density between two data streams $x(n)$ and $y(n)$ (the sampling rates of the two channels, if not already equal, must be equalized by the sample rate conversion described above) is computed as follows:

1. each data stream is segmented and de-trended & windowed as above
2. each segment of each channel is transformed with an N-point FFT
3. the FFT segment pairs are multiplied as:

$$P_{xy}(k) = \frac{1}{W}\, X(k) \cdot Y(k)^* \quad , \quad k = 0, 1, \ldots, N-1 \tag{7}$$

where * denotes complex conjugate.

4. the average over the segments is taken, as above, to form $\langle P_{xy}(k) \rangle$

3.1.6 Transfer function estimates

The transfer function estimate given an input channel *x* and an output channel *y* is the quotient of the cross-spectrum of *x* and *y* and the power spectrum of *x*:

$$\langle T_{xy}(k)\rangle = \frac{\langle P_{xy}(k)\rangle}{\langle P_{xx}(k)\rangle} \quad , \quad k = 0, 1, \ldots, \frac{N}{2} \tag{8}$$

3.1.7 Coherence estimates

The coherence between two signal vectors *x* and *y* is computed as:

$$\begin{aligned} \langle C_{xy}(0)\rangle &= \frac{\left|\langle P_{xy}(0)\rangle\right|^2}{\langle P_{xx}(0)\rangle \cdot \langle P_{yy}(0)\rangle} \\ \langle C_{xy}(k)\rangle &= \frac{\left[\left|\langle P_{xy}(k)\rangle\right|^2 + \left|\langle P_{xy}(N-k)\rangle\right|^2\right]}{\langle P_{xx}(k)\rangle \cdot \langle P_{yy}(k)\rangle} \quad k = 1, 2, \ldots, \left(\frac{N}{2} - 1\right) \\ \langle C_{xy}(N/2)\rangle &= \left|\langle P_{xy}(N/2)\rangle\right|^2 \end{aligned} \tag{9}$$

3.2 SWEPT SINE MEASUREMENTS

3.2.1 Digital demodulation & frequency response calculation

Consider a signal *s(t)* which contains a sine wave of frequency ω, of which we wish to know the amplitude; expressed as a Fourier series:

$$s(t) = \sum_{n=-\infty}^{\infty} c_n \cdot e^{i\omega n t} \tag{10}$$

The desired coefficient is calculated using the integral

$$c_1 = \frac{1}{T}\int_0^T s(t) e^{-i\omega t} dt \tag{11}$$

where $T = 2n\pi/\omega$ (i.e., an integral number of cycles). We only have a sampled version of *s(t)*, of course, which complicates the integration over an integral number of cycles; this is addressed in the next section. The steps in making a sine detection and frequency response calculation are:

1. For the desired channel, obtain the number of data points required by the integration routine (see below).
2. Multiply the data set by $e^{-i\omega t}$.
3. Integrate the real and imaginary parts of (2) over an integral number of sine wave cycles, using the numerical integration algorithm (see below), to compute the complex amplitude c_1.
4. Repeat steps 1-3 for the desired number of averages (user-specified).

5. The values of c_1 are averaged to form $\langle c_1 \rangle$; this can be used to compute the (complex) frequency response with other channels. The individual coefficients c_1 from each of the average measurements are also stored; these are needed to compute the coherence with other channels (see below).

3.2.2 Numerical integration algorithm

The numerical integration of equation (11) is done with a modified Newton-Cotes method, as follows. We find the fifth-order polynomial that passes through six adjacent points of the demodulated data series (real and imaginary treated separately), and integrate this polynomial over the middle two points. We then move forward one sample, and find and integrate the new polynomial; this process is repeated until the end of the data set, the sum of all the steps giving us the desired integral.

The last integration step must be modified to end the integration at an exact integral number of sine wave cycles. This is done by again finding the fifth-order polynomial that passes through the last six points, but only integrating over a portion of the span covered by the middle two points, stopping where a total span of an integral number of cycles has been reached.

Such an integration step can be reduced to 6 coefficients which multiply the 6 adjacent data points to give the integral over the middle two points; the sum of these 6 coefficients is unity (try integrating a constant function f(x) = 1 using this method). Thus when the whole data set is convolved with these 6 coefficients, nearly all data points are simply multiplied by unity; only 5 points at the beginning and 6 points at the end are multiplied by non-unity coefficients. Note that for each different detection frequency or integration time, the trailing coefficients must be recomputed (the lead coefficients are always the same).

This integration algorithm is similar to Simpson's rule, but uses a higher order approximation for increased accuracy. The accuracy is tested by integrating

$$\frac{1}{T}\int_0^T e^{-in\omega t}\,dt \tag{12}$$

which should integrate to zero. This is shown in Fig. 7, as a function of the number of data points in a cycle. As this shows, the method loses accuracy as the frequency increases (fewer points per cycle). To further improve the accuracy, we also pass the demodulated data through an FIR low-pass filter, prior to integration. We use a 20-tap FIR filter, with a cut-off frequency of 0.1×(Nyquist frequency). In fact, the convolution of this filter with the data can be combined with the integration operation, giving a long coefficient vector that is simply dotted (dot-product) with the data. Most of the coefficients in this vector are unity, with 24 non-unity lead coefficients and 25 non-unity trailing coefficients (the latter must be computed for each detection frequency).

The number of data points used by the integration algorithm, per average, is given by

$$n_{\mathrm{pt}} = N_{\mathrm{lead}} + 1 + \mathrm{floor}(t_{\mathrm{int}} \cdot f_s) \tag{13}$$

where $N_{\mathrm{lead}} = 24$ is the number of leading coefficients, t_{int} is the integration time (corresponding to an integral number of cycles of the sine wave), f_s is the sampling frequency, and 'floor(x)' gives the largest integer not greater than x.

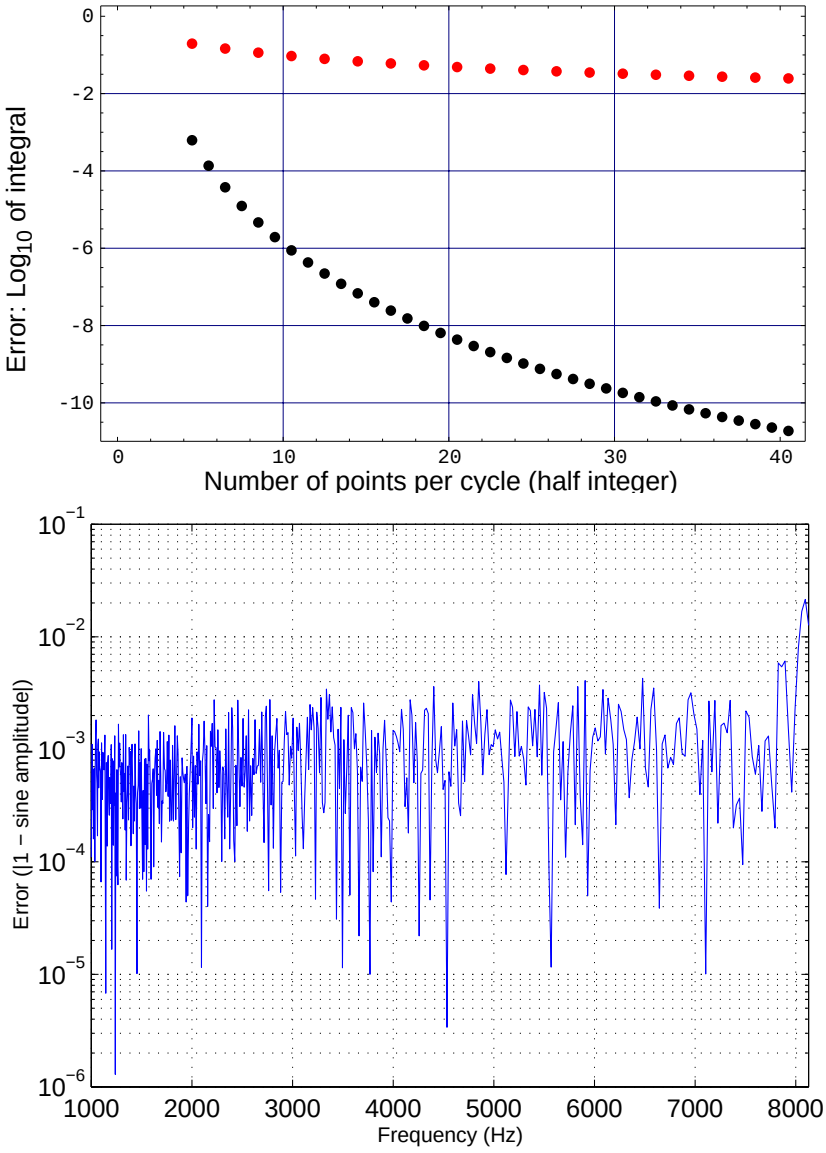


Figure 7: Accuracy of numerical integration algorithm. **Top:** plotted is the integral of equation (12) as a function of the number of points in a cycle, using Simpson's rule (modified to work over a fraction of the last span) (top/red), and the modified 5th-order polynomial method (bottom/ black). The points are all computed using a half-integral number of points in a cycle, since this is where the error is the largest. **Bottom:** Error in the integration of a unit amplitude sine wave, plus: a DC offset of 100; uniformly distributed random values in the range 0-0.1. The sine wave is sampled at 16384 Hz, and each frequency is detected with 2 averages of 100 cycles. The error increase above ~7.7 kHz because there are very few data points per cycle for the integration algorithm; below 7.7 kHz the error is due to the random values (noise) added to the data.

3.2.3 Integration time

The integration time may be specified either as a time duration, or in the number of sine wave cycles. If specified as a time interval, it is checked that it equals an integral number of cycles of the sine detection frequency; and if not, it is set equal to the nearest cycle number. For detection frequencies near a channel's Nyquist frequency, we want to ensure that a sufficient number of cycles are integrated over, and so we set a minimum integration time that is equal to 10 cycles at the Nyquist frequency (1.2 msec for a 16384/sec sample rate).

3.2.4 Settling time

At each new frequency in a swept sine test (or a sine response test), the test channels are allowed to 'settle' before a sine detection is performed on the data. The settling time is nominally 10% of the integration time. See also section 4.4.

3.2.5 Coherence calculation

Given a pair of channels on which multiple-average sine detection has been performed, the coherence between these two channels can be calculated as follows:

$$\frac{\left|\langle c_1 \cdot d_1{}^* \rangle\right|^2}{\langle |c_1|^2 \rangle \langle |d_1|^2 \rangle} \tag{14}$$

where $\{c_1\}$ and $\{d_1\}$ are the sets of complex amplitudes resulting from n sine detections on each of the two channels, and the averaging is performed over these n values.

3.3 Pole-Zero Curve Fitting

TBD

3.4 Correlation Measurements

TBD

3.5 Time Measurements

TBD

3.6 Octave Band Analysis

TBD

4 TEST ORGANIZATION

Fig. 8 shows the interactions between a diagnostics test and the application program interfaces (APIs) which communicate with the user (storage API), the network data server (Real-Time Data Distribution API) for obtaining channel data, the excitation engine (excitation API) and the test point manager (test point API).

4.1 INTERFACES

4.1.1 Storage API

All data—including test parameters, raw channel data, result arrays and plot settings—are managed by the diagnostics storage object. Any data object can be accessed by its name. Typically, the user will set test parameters prior of starting a test. The diagnostics test program will retrieve these parameters and setup the test. While the measurement is under way channel data is received by the real-time data distribution interface and automatically saved in the storage

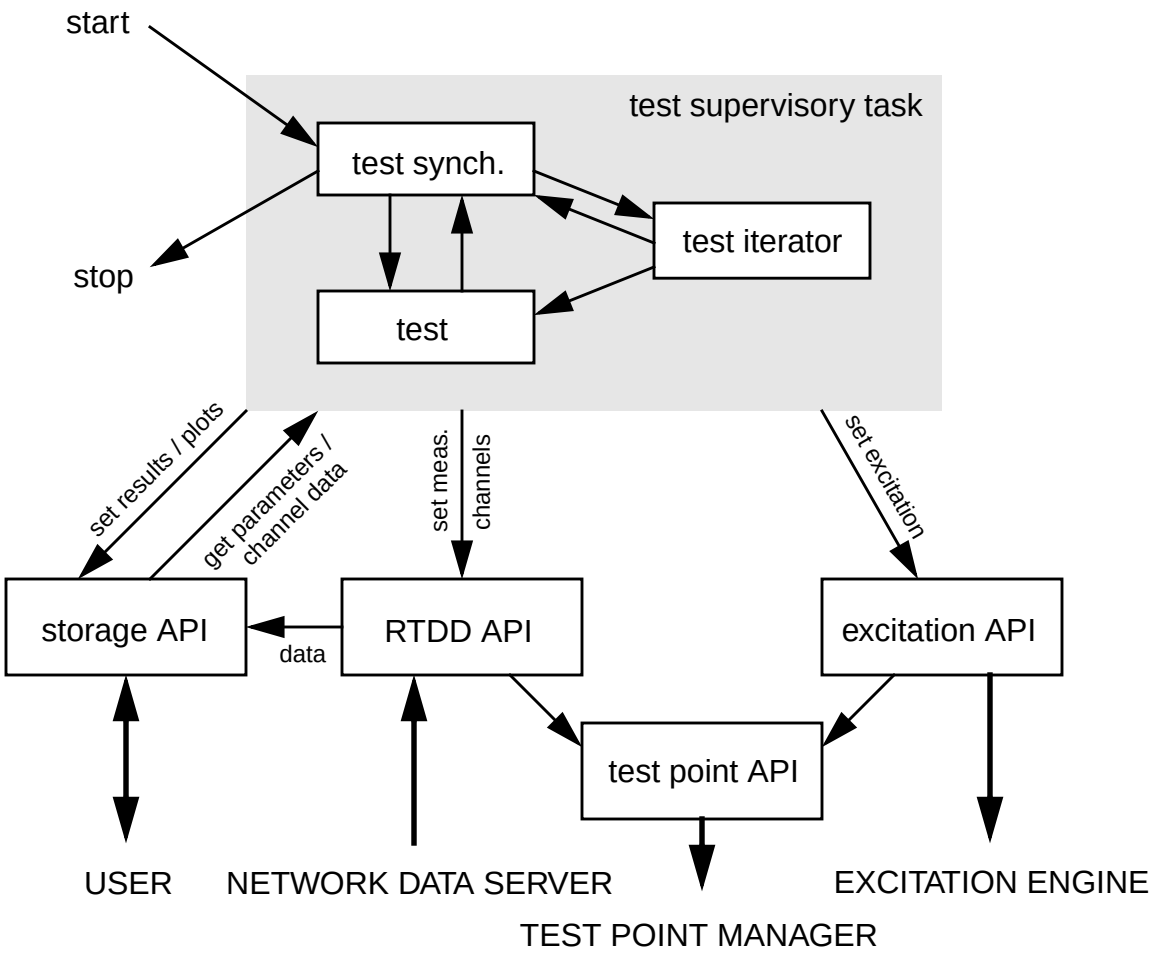


Figure 8: Interaction between a diagnostics test supervisory task and the interfaces to the real world.

object. After a measurement cycle has finished the analysis task will read the channel data, analyze it and write the results back.

4.1.2 Real-Time Data Distribution API

The real-time data distribution interface obtains data from the network data server on a continuous basis. It will subscribe the channels which are needed for the test analysis at the beginning of a test. While the test is running the data comes at a fixed rate of 16 Hz. After optional filter-decimation and down-conversion stages the data is reformatted into time slices which are useful for the analysis algorithms. Finally, the channel data is stored in the storage object.

4.1.3 Excitation API

The excitation API communicates with the excitation engine for applying excitation signals to the instrument. For a description of the excitation engine see Section 5.

4.1.4 Test Point API

The test point API communicates with the test point manager which is able to activate digital test points in the data acquisition system. Digital test points are used to read auxiliary channels from the ISC digital servo systems and to inject digital waveforms. Test points are automatically selected by the RTDD API or the excitation API if the requested channel is accessible through a test point.

The test point API also enables and disables analog excitation inputs. (Most analog inputs have an enable/disable switch which is controlled through an EPICS channel.)

4.2 TEST SUPERVISORY

When a user starts a diagnostics test, the diagnostics kernel spawns a new test supervisory task. The supervisory task is selected by setting the variable “Supervisory” to the name of the supervisory task (see Appendix B.1.1). Currently, only one supervisory task with name “standard” is supported.

4.2.1 Standard Supervisory Task

In principle every test could be implemented as its own supervisory task. This has the disadvantage that parameter scans and optimizations would need an additional supervisory task for each test they can be combined with. Also, every test would have to implement its own synchronization means. The standard supervisory task provides a solution for this problem by separating the supervisory task into a synchronization task which is common to all tests, a test iterator and the test itself. Both the test iterator and the actual test are selectable by the user. This approach has the advantage that if any new test iterator is developed, it is immediately applicable to all tests; and if a new test is developed it can use all common test iterators. Fig. 9 presents the flow chart of the standard supervisory task.

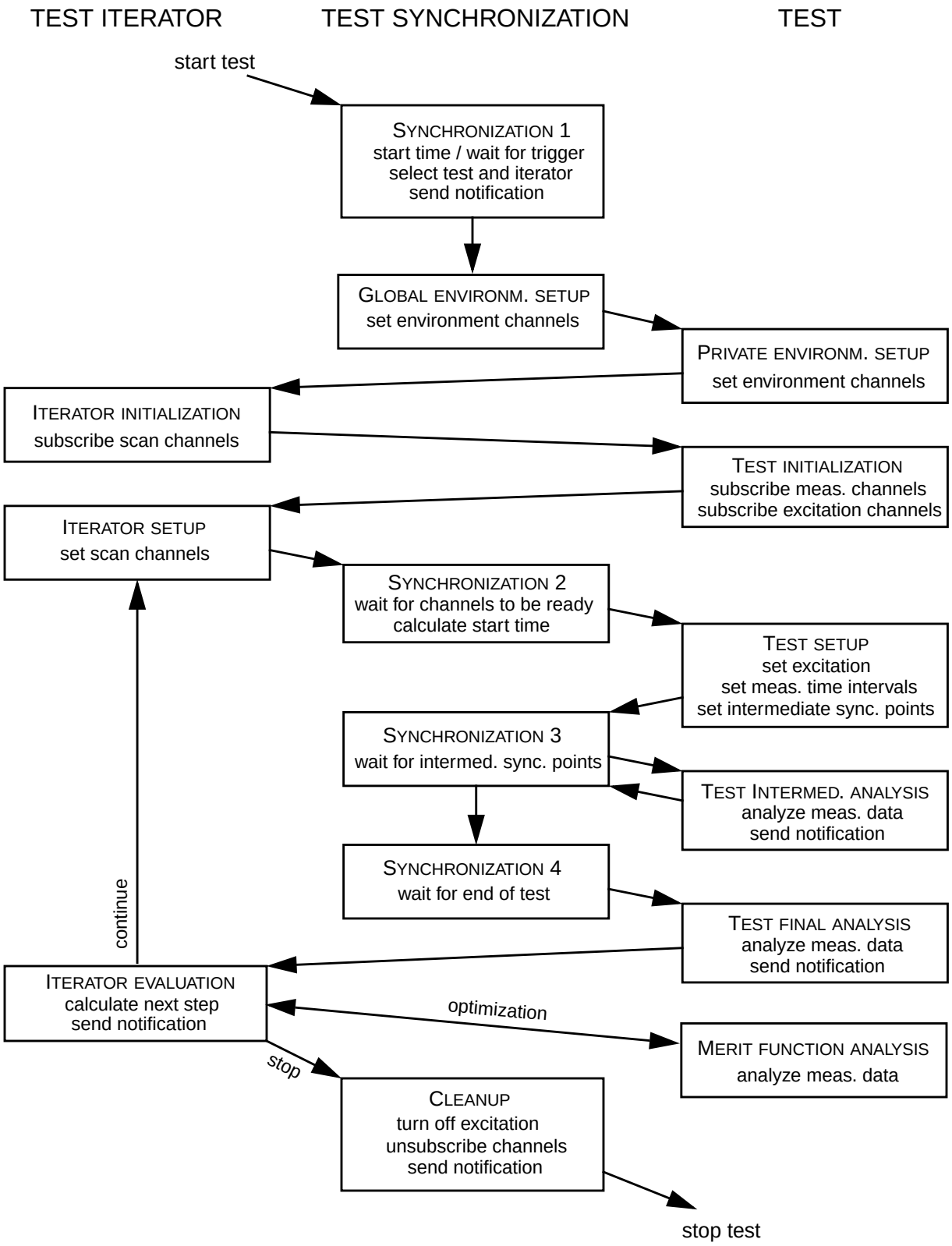


Figure 9: Flow chart of the standard diagnostics test supervisory task.

The standard supervisory task implements a few basic synchronization tools:

i) the start time can be set using the variable “Sync.Start” (GPS nsec); default is immediate.
ii) a wait time can be specified with “Sync.Wait” (sec).
iii) the execution of a test or a test step can be triggered by an EPICS channels with “Sync.WaitForStart” and “Sync.WaitAtEachStep”, respectively.
iv) a test can send a trigger signal on an EPICS channels after it or one if its steps has finished; “Sync.SignalEnd” and “Sync.SignalEndOfStep”, respectively.

At initialization the supervisory task will setup the measurement environment as specified by the “Env[N]” variable sets (see also Appendix B.1.3).

A test can be aborted at any of the synchronization steps shown in Fig. 9. Currently, a the test can only be paused after an iterator evaluation has been completed.

4.3 TEST ITERATORS

The purpose of test iterators are to repeat a standard test. In the simplest case the test is repeated with identical parameters. When performing a parameter scan the test is repeated while changing the value of a parameter. A parameter scan can be combined with an optimization to find the “best value” of a parameter set. The test iterator is selected by the variable “TestIterator” (see Appendix B.1.1).

4.3.1 Repeat

The default test iterator is the trivial one with a repetition value of one. The number of repeats can be set with the variable “Sync.Repeat”, whereas the repeat rate is determined by “Sync.RepeatRate”. This repeat rate is interpreted as a minimum time between tests. In other words the second test is not started before the first one has finished completely.

4.3.2 Parameter Scan

For a parameter scan at least one “Scan[N]” variable sets have to be defined and set active (see Appendix B.1.4). A test will be repeated while scanning a parameter of an excitation channel such as the amplitude, the offset or the frequency. An example would be to investigate the coupling of input beam jitter into the gravitational wave band (sine response test) while changing the angular alignment of the interferometer test masses.

4.3.3 Optimization

For an optimization to take place the “Find” variable set has to be defined and enabled. Additionally, a valid parameter scan has to be setup (see Appendix B.1.5). An optimization is a parameter scan where parameters are adjusted to their “best value”. Typically, the parameter is set to a new value after a scan interval has been completed. The “best value” is determined by a merit function which has to be provided by the test.

4.4 TESTS

4.4.1 Sine Response

The timing diagram of a sine response test is shown in Fig. 10. The full test time is divided into a dwell time and a number of measurement periods. The dwell time is divided into a ramp up period and a settling time (see also Appendix B.3.1). The settling time, t_S, is determined by the "Test.SettlingTime" values, t_Δ and n_c, according to the following formula (negative values are ignored):

$$t_S = \min\left(t_\Delta, \frac{n_c}{f}\right) \tag{15}$$

where f represents the frequency of the excitation signal. In case of multiple excitations with difference frequencies the smallest frequency value is used for the calculation.

During the ramp up time the amplitude of the excitation signals are slowly increase to their final requested levels. The ramp up time, t_R, is determined by the following formula:

$$t_R = \min(t_S, t_{\max}) \tag{16}$$

where $t_{\max} = 1\,\text{sec}$ is the maximum allowed ramp up time. The total dwell time $t_R + t_S$ is always rounded up to the next sampling period. If an excitation signal can not be synchronized with a

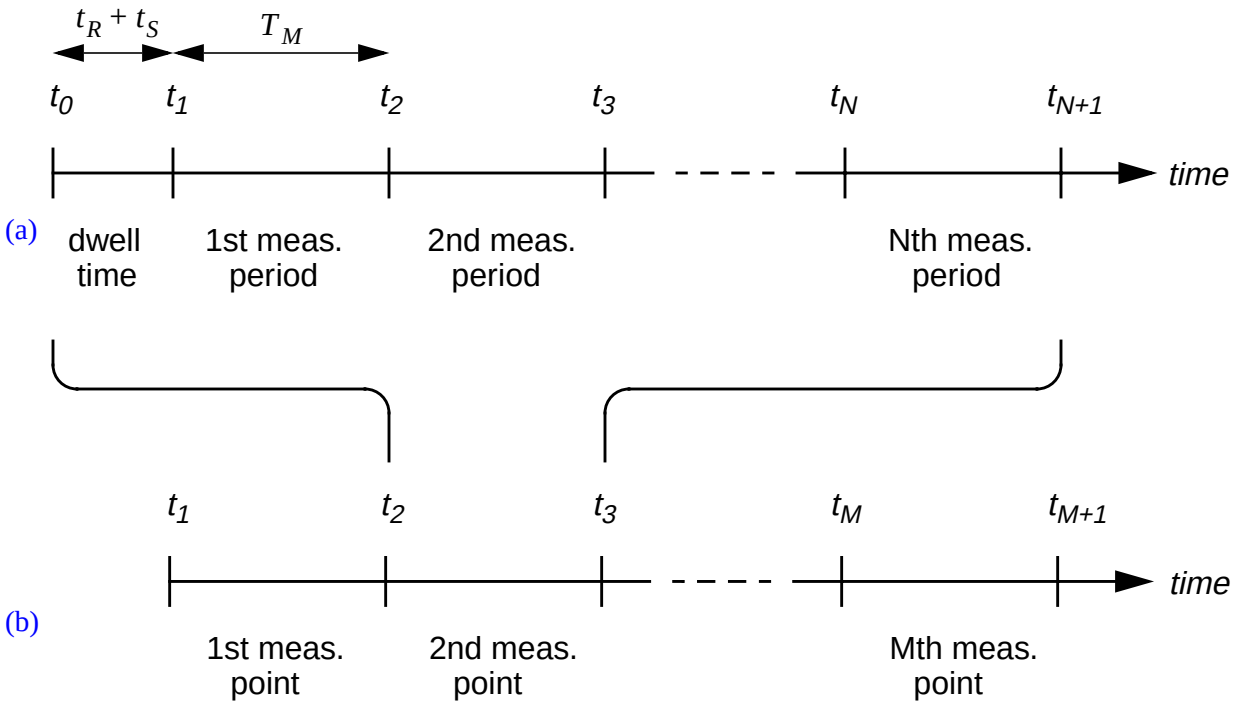


Figure 10: Timing diagram of a sine response (a) and a swept sine test (b). *N* and *M* are the number of averages and the number of measurement points, respectively.

GPS clock, the minimum ramp up time is set to 500 ms to account for network latencies. The default ramp up function is a quadratic phase-in, see Eqns. (34) and (37).

The time of a single measurement T_M is determined by the "Test.MeasurementTime" values, T_Δ and N_c, according to the same formula as the settling time with the only modification that T_M is always rounded up the next full cycle and sampling period. If multiple excitation frequencies are present, the smallest frequency is used to perform the calculation.

4.4.2 Swept Sine

A swept sine is essentially a series of sine response tests with a single excitation frequency which is swept through a predefined frequency interval. The frequency points can be spaced linearly, logarithmically or by user supplied values. The frequency points f_i are calculated from the start and stop frequencies, t_{start} and t_{stop} with the following formulae:

linear
$$f_i = f_{\text{start}} + (i - 1)\frac{f_{\text{stop}} - f_{\text{start}}}{M - 1} \tag{17}$$

logarithmic
$$f_i = f_{\text{start}}\left(\frac{f_{\text{stop}}}{f_{\text{start}}}\right)^{\frac{i-1}{M-1}} \tag{18}$$

where M represents the number of frequency points and the index i runs from 1 to M.

The dwell time is calculated the same way is for a simple sine response test. The only difference is that the ramp up signal between measurement points is replaced by a ramp signal which guarantees a smooth transition in both amplitude and frequency. For the amplitude Eqns. (34) and (37) are used again; for sweeping the frequency Eqns. (38) and (39) are used for linear swept sine tests and Eqns. (40) and (41) for logarithmic swept sine tests, respectively.

4.4.3 FFT Tests

The timing diagram of an FFT test is shown in Fig. 11. The total test time is broken into possibly overlapping measurement intervals. The number of sampling points for each measurement period is always set to a power of 2. The time of a single measurement interval can be calculated from the specified bandwidth f_{BW} by:

$$T_M = \frac{1}{f_{BW}} \tag{19}$$

Since the measured time series is down-sampled to a sampling rate of twice the frequency spawn of the FFT,

$$f_{\text{sample}} = 2(f_{\text{stop}} - f_{\text{start}}) \tag{20}$$

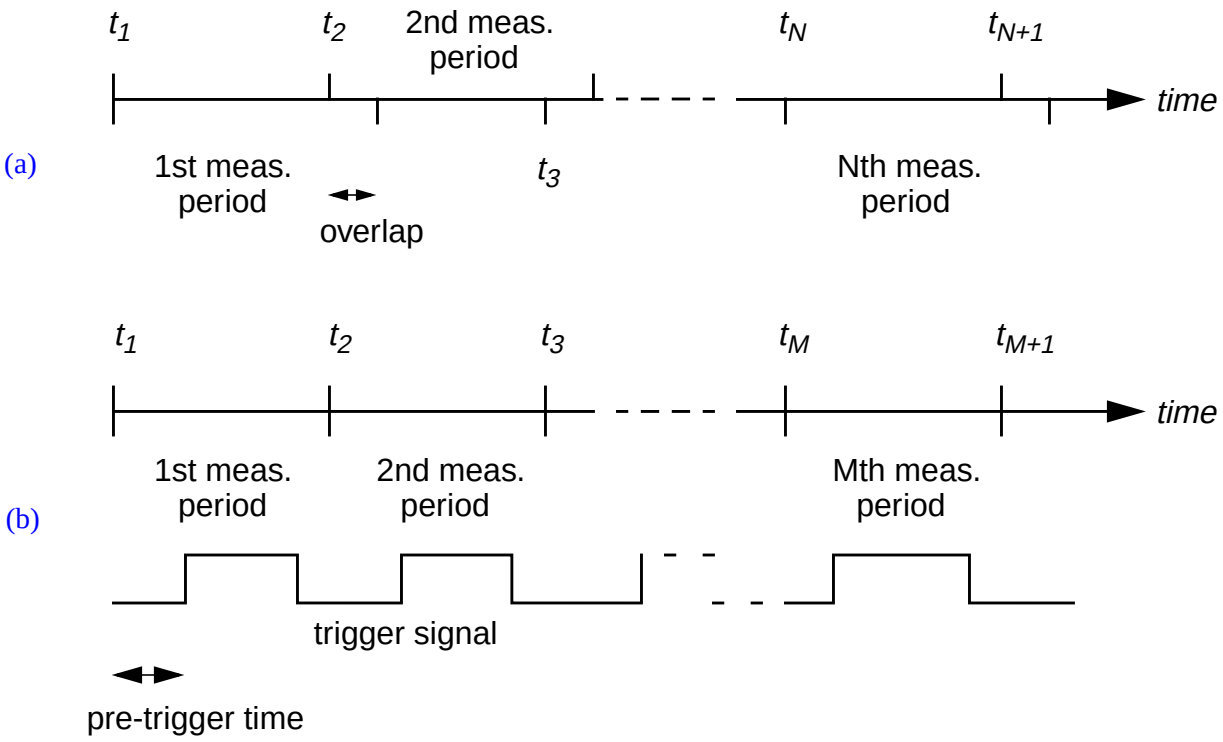


Figure 11: Timing diagram of an FFT test (a) and a time series/trigger response test (b). *N* and *M* are the number of averages and the number of measurement points, respectively.

the number of points sent to the FFT algorithm can be written as:

$$N_{FFT} = T_M \, f_{\text{sample}} = 2 \, \frac{f_{\text{stop}} - f_{\text{start}}}{f_{BW}} \tag{21}$$

Of course, if a non-uniform window function is selected, the effective measurement bandwidth is determined by the width of the window function, i.e.

$$f_{BW}^{\text{eff}} \approx \frac{1}{\Delta t_{\text{window}}} \tag{22}$$

If the number of averages is greater than 1, the start time t_i for each measurement interval is given by

$$t_i = t_o + i(1-\eta)T_M \tag{23}$$

with η the overlap factor.

4.4.4 Time Series Measurements

Time series and trigger response measurements are broken up in equal length measurement intervals (see Fig. 11) which is related to the trigger rate by

$$T_M = \frac{1}{f_{\text{trigger}}} \tag{24}$$

In order to keep the averaging algorithms simple the length of measurement interval has to be a multiple of the sampling period. Or in other words, the trigger rate has to be an integer fraction of the sampling frequency. The pre-trigger time specifies the time the start of the trigger signals lags behind the start of the measurement.

4.4.5 Random Stimulus Response Test

TBD.

5 EXCITATION ENGINE

5.1 OUTPUT WAVEFORMS

5.1.1 Overview

Every output signal produced by the arbitrary waveform generator consists of two parts: a fundamental waveform and phase-in and phase-out transition. The following fundamental waveforms are supported by the signal generator:

sine wave
$$s(t) = A\sin(2\pi ft + \phi) \tag{25}$$

square wave
$$s(t) = \begin{cases} +A & 0 \le 2\pi ft + \phi \bmod 2\pi < \pi \\ -A & \pi \le 2\pi ft + \phi \bmod 2\pi < 2\pi \end{cases} \tag{26}$$

ramp
$$s(t) = A\frac{\phi}{2\pi} \text{ with } \phi = 2\pi ft \bmod 2\pi \tag{27}$$

triangle
$$s(t) = \begin{cases} A\left(\frac{2\phi}{\pi} - 1\right) & 0 \le 2\pi ft + \phi \bmod 2\pi < \pi \\ A\left(3 - \frac{2\phi}{\pi}\right) & \pi \le 2\pi ft + \phi \bmod 2\pi < 2\pi \end{cases} \tag{28}$$

flat noise (29)

band limited noise (30)

pink noise (31)

sweep (32)

5.1.2 Periodic Waveforms

The output signal $o(t)$ of an arbitrary waveform generator for a periodic waveform starting at time t_0, lasting for a duration Δt, ramping up between t_0 and $t_0 + t_{PI}$, and phasing out after $t_0 + t_2$ ($t_2 = \Delta t - t_{PO}$) can be written as:

$$o(t) = s(2\pi f(t - t_0) - \phi_0 + \varphi(t - t_0 - t_2, t_{PO})) \times b(t - t_0, \Delta t, t_{PI}, t_{PO}) \tag{33}$$

where $s(\phi)$ is the periodic signal (fundamental waveform) of frequency f and phase shift ϕ_0. The functions φ and b are then functions which determine the phase-in and phase-out transitions at

the beginning and at the end of the output signal. The phase-in and phase-out times are denoted with t_{PI} and t_{PO}, respectively. The function b controls the amplitude and we write it as:

$$b(t-t_0, \Delta t, t_{PI}, t_{PO}) = \begin{cases} 0 & t<0 \text{ or } t>\Delta t \\ 1 & t_{PI} \le t < \Delta t - t_{PI} \\ g_1(t, t_{PI}) & 0 \le t < t_{PI} \\ g_2(t-t_2, t_{PO}) & \Delta t - t_{PO} \le t \le \Delta t \end{cases} \tag{34}$$

where g_1 and g_2 are the amplitude phase-in and phase-out functions, respectively. The following options are available: step, linear ramp, and quadratic.

step
$$\begin{aligned} g_1(t, t_{PI}) &= 0 \\ g_2(t, t_{PO}) &= 1 \end{aligned} \tag{35}$$

linear
$$\begin{aligned} g_1(t, t_{PI}) &= \frac{t}{t_{PI}} \\ g_2(t, t_{PO}) &= 1-(1-c)g_1(t, t_{PO}) \end{aligned} \tag{36}$$

quadratic
$$\begin{aligned} g_1(t, t_{PI}) &= -\left(\frac{t}{t_{PI}}\right)^4 + 2\left(\frac{t}{t_{PI}}\right)^2 \\ g_2(t, t_{PO}) &= 1-(1-c)g_1(t, t_{PO}) \end{aligned} \tag{37}$$

where c denotes the amplitude ratio of the signal before the phase-out and the one after. For two periodic waves of different frequencies which should smoothly transform into each other, it is usually more practical to sweep the frequency, rather than phase-out one signal by ramping it down while ramping up the other one. The sweep function φ is defined as follows:

$$\varphi(t, t_{PO}) = g_1(t, t_{PO})(2\pi \Delta f\, t - \overline{\Delta\phi}) \tag{38}$$

with $\Delta f = f_2 - f_1$ the frequency difference between the beginning and the end of the sweep. The phase adjustment $\overline{\Delta\phi}$ can be written as:

$$\overline{\Delta\phi} = \Delta\phi + 2\pi f \Delta t + 2\pi \Delta f\, t_{PO}\big|_{\text{mod } 2\pi} \tag{39}$$

with $\Delta\phi$ the phase difference between the signal at $t = t_0$ and $t = t_0 + \Delta t$.

The above sweep function can be readily generalized to logarithmic sweeps. For the time during the sweep one can rewrite Eqn. (33) as follows:

$$o(t)\big|_{t_2 \le t - t_0 \le \Delta t} = s(2\pi f t_2 - \phi_0 + \varphi_{\log}(t - t_0 - t_2, t_{PO})) \tag{40}$$

with the sweep function

$$\rho_{\log}(t, t_{PO}) = \left[2\pi f t_{PO} \left(\frac{f + \Delta f}{f} \right)^{\frac{t}{t_{PO}}} - \overline{\Delta\phi} \right] \frac{t}{t_{PC}} \tag{41}$$

where the phase adjustment is the same as in Eqn. (39).

5.1.3 Non-Periodic Waveforms

Non-periodic output waveforms can be written as:

$$o(t) = n(t - t_0) \times b(t - t_0, \Delta t, t_{PI}, t_{PO}) \tag{42}$$

with the only difference to a periodic waveform that the actual signal $n(t)$ does depend on the time directly rather than the phase, and that no sweep function is provided.

APPENDIX A MESSAGE PASSING INTERFACE

A.1 USING REMOTE PROCEDURE CALLS

The rpc message passing interface implements four basic functions:

i) open a connection to a diagnostics kernel,
ii) close the connection to a diagnostics kernel,
iii) send a message to a diagnostics kernel and wait for the answer, and
iv) install a callback function for messages send back by the diagnostics kernel.

These functions are defined in the C header file ‘gdsmsg.h’ (for a more detailed description see there, or in the corresponding web page). A message consists of a message header and a parameter argument; it returns a reply argument. Valid message headers are ASCII strings which are generally identical to the commands which are recognized by the diagnostics kernel as specified in section 2.3. In general, a command and its ASCII arguments are passed through the message header, whereas binary arguments are passed through the parameter argument.

One important difference to the command line interface are the ‘save’ and ‘restore’ commands: *(i)* the flag argument must be specified, *(ii)* the filename is assumed to be local to the machine which runs the diagnostics kernel, and *(iii)* the reserved filename specifier ‘-’ is used to indicate that file is part of the message. When using ‘save’, the file is passed back through the reply argument. When using ‘restore’ the file has to be passed to the diagnostics kernel through the parameter argument.

The diagnostics kernel can run on a local machine—in this case the shared object library ‘libgds.so.1’ is loaded dynamically when the connection is established—or it can run on a remote machine—in this case the diagnostics kernel has to run on the remote machine listening at the designated rpc port, or the it has to be added to the list of services of the inet daemon. Assuming the full path name of the diagnostics daemon is ‘/home/gds/bin/gdsd’ the following configuration line should be added to the ‘inet.conf’ file:

```
822087684/1   stream   rpc/tcp   nowait   gds   /home/gds/bin/gdsd   gdsd
```

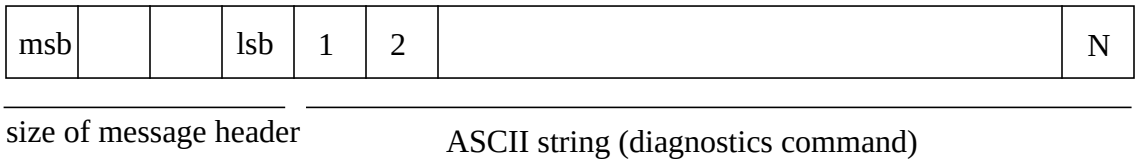
Whenever a request is made at the specified remote procedure port a new diagnostics kernel is launched and a private two way communication channel is established between the caller and the diagnostics kernel. After closing the connection the diagnostics kernel is automatically terminated.

A.2 USING SOCKETS

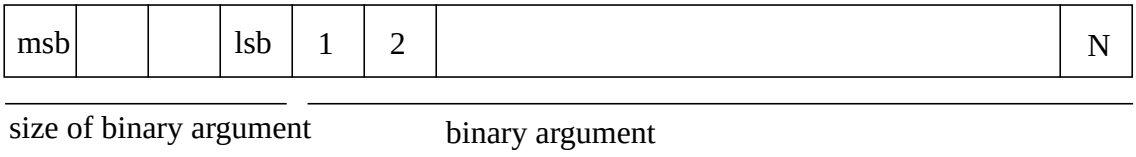
An alternative way of connecting to the diagnostics kernel is using TCP/IP sockets. After opening a connection to the diagnostics server, it will wait for a 32 bit flag (MSB first) which describes the required services. It will answer by sending back two 32 bit numbers, the first one describing the supported capabilities and the second one representing a port number which can be used to establish a connection to receive notification messages. If the socket is closed the server automatically terminates.

Messages are passed to the server using the following format:

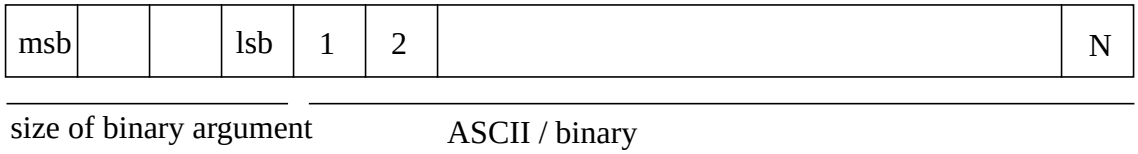
Header:



Binary argument:



Return argument:



Every message consists of a header and a binary argument (its size can be zero); it will return either an ASCII or a binary argument dependent on the command. The server will send the answers back in the order the commands were received. Notification messages are send from the diagnostics kernel to the user interface using the same format through a separate connection to avoid confusion in the main connection.

In order to listen to a TCP/IP port rather than an rpc port the diagnostics kernel must be launched with the ‘-s’ option. The diagnostics kernel can also be started by the inet daemon. Assuming the full path name of the diagnostics daemon is ‘/home/gds/bin/gdsd’ the following configuration lines should be added to the ‘inet.conf’ and the ‘services’ files, respectively:

```
gds   stream   tcp   nowait   gds   /home/gds/bin/gdsd   gdsd -s
gds   5354/tcp
```

Whenever a request is made at the specified remote procedure port a new diagnostics kernel is launched and a private two way communication channel is established between the caller and the diagnostics kernel. After closing the connection the diagnostics kernel is automatically terminated.

APPENDIX B DATA REPRESENTATION

The diagnostics kernel manages variables, parameters and data which describe a diagnostics test using the same basic representation. The basic storage objects are data objects and parameter objects. Parameter objects consists of a name, a type and a value or a list of values. A data object is a container which can contain a multidimension data array, a set of parameter objects and other data objects. The diagnostics kernel stores all variables associated with a diagnostics test within a single (global) data object. The diagnostics kernel limits the hierarchical levels of data objects to two. Meaning, the global object can contain parameters and data objects which in turn can contain other parameter objects, but no more data objects.

Every data object and every parameter object within a single data object must have a unique name. Any parameter can then be accessed by specifying its name and the name of its data object (group name). The parameters in the global data objects can be accessed by their name only, or by specifying an empty data object name. A name is not case sensitive and can not contain the '.' character, since this character is used to separate its name from the group name.

A name must not contain square brackets, because they are used for indexing. Names and group names can have an array index, e.g., 'Scan[1]' or 'H0:GDS_TEST[0][2]'. A maximum of two array indices are supported with values ranging from 0 to 999.

Both data and parameter objects have associated data fields consisting of a data type, a dimension list, the actual datum, an optional string describing the physical unit and an optional comment string. Supported data types are:

Name	C/C++ data type	Abbreviation
8 bit integer	char	c
16 bit integer	short	s
32 bit integer	int	i
64 bit integer	long long	ll
boolean	bool	b
single precision floating point	float	f
double precision floating point	double	d
single precision complex number	complex<float>	zf
double precision complex number	complex<double>	zd
string	char*	st
channel name	char*	ch

Parameter objects typically have a single value associated with them, whereas data objects can be multi-dimensional.

The diagnostivcs kernel uses a set of data objects and parameters with predefined names for describing: *(i)* the test parameters, *(ii)* the test results, *(iii)* the raw data and *(iv)* the plot options. The following data objects are used by the diagnostics kernel:

Name	Type	Description
Def	test param.	Global default settings of diagnsotics tests.
Sync	test param.	Synchronization information.
Env[N]	test param.	Excitation environment; N from 0 to 99.
Scan[N]	test param.	Scan parameters; N from 0 to 9.
Find	test param.	Optimization parameters.
Test	test param.	Specific settings for a diagnostics test.
‘channe name’[M][N]	raw data	measured time series; M and N from 0 to 999.
Result[N]	result	results of diagnostics test: FFT, transfer functions, averaged time series, coefficients and list of measurement values; M from 0 to 999.
Plot[N]	plot settings	result plots; M from 0 to 99.

All data objects have a string parameter of name ObjectType which identifies the data object. The following object types are supported:

ObjectType	Associated data objetcs
DiagnosticsTest	global
Defaults	Def
Synchronization	Sync
Environment	Env[N]
Scan	Scan[N]
Optimization	Find
TestParameter	Test
TimeSeries	‘channe name’[M][N] or Result[N]
Spectrum	Result[N]
TransferFunction	Result[N]
Coefficients	Result[N]
MeasurementTable	Result[N]
Plot	Plot[N]

B.1 COMMON

This section list the parameters which are common to all diagnostics tests.

B.1.1 GLOBALS

A few parameters are defined in global scope:

Name	Type	Dim	Description
ObjectType	st	1	DiagnosticsTest
TestType	st	1	describes the diagnostics test class. Possible values are: TimeSeries, SweptSine, FFT, SineResponse, RandomResponse.
TestName	st	1	name of test, user supplied.
Supervisory	st	1	name of supervisory task; default standard.
TestIterator	st	1	name of test iterator; default repeat. Possible values are: repeat, scan, find.
Comment	st	1	user supplied comment.
TestTime	ll	1	time when test was done in GPS nsec.
TestTimeUTC	st	1	time when test was done in UTC format, e.g. 1998-11-08 17:40:00.032035.

The following default parameters are common to all tests:

Name	Type	Dim	Description
Def.ObjectType	st	1	Defaults
Def.AllowCancel	b	1	if true (default), cancel a test is allowed at any time.
Def.NoStimulus	b	1	if true, no stimulus is applied; default is false.
Def.NoAnalysis	b	1	if true, omits the analysis; default is false.
Def.SiteDefault	c	1	describes the default site; any channel name that contains an ‘X’ as the site identifier will be adjusted to the specified default site.
Def.SiteForce	c	1	overrides the site identifier in every channel name.
Def.IfoDefault	c	1	describes the default interferometer number. Any channel name containing an ‘X’ instead of an interferometer number will be adjusted to the specified interferometer.
Def.ifoForce	c	1	overrides any none zero interferometer identifier in every channel name.

Channel names can be specified site and interferometer indepdendent. To do so the character ‘X’ is used instead if the site or the interferometer identfier, respectively. Any ‘X’ character will be replaced with its corresponding default value before the test is started. Alternatively, it is possible

to override site or interferometer identifiers of all channel names. This can be useful when moving a diagnostics test from one instrument to an other.

B.1.2 SYNCHRONIZATION TOOLS

Diagnostics test can be synchronized with EPICS channels. A test can halt at the beginning of a test and/or at each step of a measurement until a trigger signal is received on a specified EPICS channel. Similarly, a diagnostics test can send a trigger signal on a specified EPICS channel at the end of a test and at the end of each measurement step.

Name	Type	Dim	Description
Sync.ObjectType	st	1	Synchronization
Sync.Start	ll	1	start time of test (GPS nsec); default 0 (now)
Sync.Wait	d	1	time to wait before starting test (in sec); default 0. the wait time is added to the start time.
Sync.Repeat	i	1	number of times of repeating the test; default 1.
Sync.RepeatRate	d	1	rate of repeating tests (in sec); default 0 (no wait).
Sync.WaitForStart	ch	1	if defined, the test only starts after receiving a trigger signal on the specified EPICS channel (waits for a 1; resets channel to zero after trigger was received).
Sync.WaitAtEachStep	ch	1	if defined, the test only starts a new step after receiving a trigger signal on the specified EPICS channel.
Sync.SignalEndOfStep	ch	1	if defined, signals the end of a test step on the specified EPICS channel (sets channel to 1).
Sync.SignalEnd	ch	1	if defined, signals the end of the test on the specified EPICS channel.

B.1.3 ENVIRONMENT

While performing a diagnostics test, excitation and EPICS channels can be set to specified waveforms and values, respectively. This environment is set before the test starts and automatically reset after the test terminates. Environments are numbered, describing a single excitation channel each:

Name	Type	Dim	Description
Env[N].ObjectType	st	1	Environment
Env[N].Active	b	1	if true (default), includes the Env[N] excitation channel.
Env[N].Channel	ch	1	string describing an excitation channel. The format is: ‘channel name’ ‘waveform’ where waveform is of the same format as described in Section 2.3.2.

Name	Type	Dim	Description
Env[N].Waveform	st	1	string describing an excitation waveform. The format is the same format as described in Section 2.3.2.
Env[N].Points	f	M	describes the values of an arbitrary waveform.
Env[N].Wait	d	1	time to wait for the environment to settle down.

If an excitation channel is defined in the environment and is also used by a stimulus response test, the two excitation waveforms are simply added. Certain restrictions apply if the excitation channel is a stand-alone signal generator.

B.1.4 PARAMETER SCAN

Any test can be repeated while sweeping either frequency, amplitude or offset of one or multiple excitation channels. Sweep parameters are numbered from N = 0, 2, ... 9, describing a single excitation channel each:

Name	Type	Dim	Description
Scan[N].ObjectType	st	1	Scan
Scan[N].Active	b	1	if true (default), includes Scan[N] in the sweep.
Scan[N].Channel	ch	1	name(s) of sweep channel, can be an excitation channel, or an EPICS analog output.
Scan[N].Type	i	1	0 – linear sweep (default), 1 – logarithmic sweep, 2 – user supplied sweep points.
Scan[N].Direction	i	1	0 – upwards (default), 1 – downwards.
Scan[N].Parameter	i	1	0 – offset (default), 1 – amplitude, 2 – frequency Only offset is valid for an EPICS channel.
Scan[N].Frequency	d	1	specifies the frequency of the signal (not used, if the frequency is swept, or if EPICS channel).
Scan[N].Amplitude	d	1	specifies the amplitude of the signal (not used, if the amplitude is swept, or if EPICS channel).
Scan[N].Offset	d	1	specifies the offset of the signal (not used, if the offset is swept, or if EPICS channel).
Scan[N].Start	d	1	specifies the start point of the sweep (not used when user supplies sweep points).
Scan[N].Stop	d	1	specifies the stop point of the sweep (not used when user supplies sweep points).
Scan[N].N	i	1	specifies the number of points of the sweep (not used when user supplies sweep points).
Scan[N].Points	f	M	specifies the user supplied sweep points
Scan[N].Wait	d	2	specifies the settling time before each measurement; default is 10 cycles, or 65 ms, whatever is shorter. first value is time in sec.

The dimension of the scan is determined by the number of defined channel names. If fewer than N values are specified for an N dimensional parameter, the remaining values are automatically padded with zero.

A multi-dimensional linear or logarithmic sweep will use a multi-dimensional mesh to set the sweep points. For a user defined multi-dimensional sweep the user has to supply every sweep point which will be used; they can be located anywhere in the multi-dimensional parameter space.

If the same excitation channel which is defined in the environment and/or by a stimulus response test, is also used by a sweep, the excitation waveforms are simply added. Certain restrictions apply if the excitation channel is a stand-alone signal generator

B.1.5 PARAMETER OPTIMIZATION

A parameter scan can be combined with a parameter optimization. When optimization is enabled the sweep will try to set the sweep channel values to the best values as determined by a merit function. For a multi-dimension linear or logarithmic sweep the optimization process will scan through each dimension separately, and set the signal to the best value after each scan. A multi-dimensional sweep with user supplied sweep points is interpreted as a one dimensional optimization along the path of the user supplied points.

Name	Type	Dim	Description
Find.ObjectType	st	1	Optimization
Find.Enable	b	1	if true, enables the optimization process; must be selected together with a sweep. Default is false.
Find.Change	b	1	if true (default), the detector state at the end of the test will be set back to the values before the test, i.e. the optimization process tries to find an optimal parameter set without changing to the new and better state.
Find.Type	i	1	0 – maximum, 1 – minimum, 2 – zero, 3 – value
Find.Value	d	1	value to find, if type is ‘value’.
Find.Function	i	1	The merit function which is used depends on the selected diagnostics test: SineResponse (single frequency only): 0 – amplitude, 1 – phase, 2 – harmonic distortion, 3 – intermodulation product (added in quadrature) TimeSeries: 0 – average, 1 – rms FFT or RandomResponse: 0 – band-limited power
Find.Param	d	2	low and high frequency values for band-limited power
Find.Method	i	1	0 – scan

B.2 RESULTS

A diagnostics test can consist of multiple measurements such as the individual steps of a sweep. Each measurement can contain multiple measurement points, such as the frequency points of a swept sine measurement. Typically, there is the need to store results of individual measurement points, as well as the results of each measurement step and the final result. This is achieved by using array indices which are part of the result name. However, there is no one-to-one relation between result index and measurement step or point. Instead, each result data object stores the results of a whole measurement step and encodes the step number (if not a final result) in the comment string using the C format ‘step %i’. Generally, plots of intermediate steps are not recorded, but rather renewed whenever a new value is measured.

B.2.1 PLOT

A plot object does not contain the data it displays, rather it describes the settings of the plot and stores pointers to the data objects which contain the plotted data. To limit the number of result objects, a plot object can handle A and B channels. When displaying a transfer function or a cross-power spectrum, A and B channels point to the individual measurement channel; and the plotting routine is responsible to calculate the ratio or product, respectively. The following options are used to control the appearance of the plot.

Parameter Name	Type	Dim	Description
ObjectType	st	1	Plot
PlotType	i	1	0 – XY plot (line/scatter), 1 – time series (line/scatter), 2 – power spectral density (PSD), 3 – Bode plot (dB/phase plots, line), 4 – histogram (lego). default: 0.
Trace	i	16	0 – none 1 – A channel 2 – B channel 3 – cross spectrum (PSD) 4 – transfer function B/A (PSD, Bode) 5 – coherence (PSD, Bode) 6 – normalized variance (PSD, Bode) default: 1.
PlotLabel	st	1	Plot label. Use TEX convention for greek letters, superscript and subscript, e.g. ‘α_1’ default: empty string (no label).
AspectRatio	d	1	aspect ratio of plot. default: 0.7.
Frame	b	1	if true, plot is framed. default: true.
Label[N]	st	1	Frame labels, numbering: clockwise starting at lower x-axis. default: empty strings (no labels).

Parameter Name	Type	Dim	Description
Font	i	1	0 – Helvetica, 1 – Times. default: 0.
FontSize	d	3	Font sizes of plot label, frame labels and axes label, respectively (in relative units to the diagonal plot length). default: 0.045, 0.04, 0.03.
Axes	b	2	whether to draw x-axis (first value) and y-axis (second value). default: true, true.
AxesType	i	2	0 – linear, 1 – log. default: 0, 0.
AxesPlacement	d	1	place of axes (in relative units to the diagonal plot length); positive unit – axes are placed outside default: 0.05.
AxesThickness	d	1	tickness of axes (in relative units to the diagonal plot length). default: 0.004.
AxesLabel	i	2	0 – no axes labels, 1 – labels at major ticks, 2 – labels at major and minor ticks. default: 1, 1.
MajorTicks	i	2	0 – automatic, 1 – no major ticks, 2 – user specified. default: 0.
MajorTicksStyle	i	2	0 – both directions, 1 – inside only, 2 – outside; default: 1, 1.
MajorTicksLength	d	2	length of major ticks (in relative units to the diagonal plot length). default: 0.03, 0.03.
TicksX	d	M	where to place ticks (x-axis)
TicksY	d	M	where to place ticks (y-axis)
MinorTicks	i	2	0 – automatic, 1 – no minor ticks, >1 – number of minor ticks intervals. default: 0.
MinorTicksStyle	i	2	0 – both directions, 1 – inside only, 2 – outside. default: 1, 1.
MinorTicksLength	d	2	length of major ticks (in relative units to the diagonal plot length). default: 0.015, 0.015.
GridLines	i	2	0 – no grid lines, 1 – dotted grid lines at major ticks, 2 – dashed grid lines at major ticks, 3 – dashed grid lines at major ticks, dotted at minor. default: 1, 1.

Parameter Name	Type	Dim	Description
PlotRange	i	2	0 – automatic, 1 – user specified. default: 0, 0.
PlotRangeLimits[N]	d	2	lower and upper limits of plot range. default: 0, 1.
BinType	i	1	0 – no binning, 1 – averages values of bins 0, ... Bin–1 into bin 0, 2 – averages values of bins 0, ... Bin–1 into (Bin–1)/2, 3 – adds values of bins 0, ... Bin–1 into bin 0, 4 – adds values of bins 0, ... Bin–1 into (Bin–1)/2. default: 0.
Bin	i	1	number of bins. default: 1.
MarkerType	i	1	0 – no marker, 1 – x-axis markers, 2 – y-axis markers, 3 – trace markers. default: 0.
Marker	d	2	Marker values. default: lower and upper plot range.
PlotStyle	i	16	0 – lines, 1 – symbols, 2 – lines and symbol. default: 0.
LineColor	i	16	–1 – automatic 0 – black, 1 – red, 2 – blue, 3 – green, 4 – yellow, 5 – brown, 6 – cyan, 7 – purple, default: –1 (0 on paper; 0, ... 7 on screen)
LineType	i	16	–1 – automatic 0 – solid 1 – dashed 2 – doted 3 – dash-doted 4 – dash-dot-doted 5 – dash-dash-doted 6 – dash-dash-dot-doted 7 – long dashed default: –1 (0 on screen, 0, ... 7 on paper)
LineTickness	d	32	tickness of lines (in relative units to the diagonal plot length). default: 0.004.

Parameter Name	Type	Dim	Description
SymbolShape	i	16	–1 – automatic 0 – circle, 1 – triangle up, 2 – square, 3 – triangle down, 4 – diamond, 5 – star, 6 – cross, 7 – slanted cross. default: –1 (0, ... 7).
SymbolSize	d	1	size of symbols (in relative units to the diagonal plot length). default: 0.02.
ErrorBars	i	16	0 – no error bars, 1 – symmetrical vertical error bars, 2 – symmetrical horizontal error bars, 3 – symmetrical vertical and horizontal error bars, 4 – asymmetrical vertical error bars, 5 – asymmetrical horizontal error bars, 6 – asymmetrical vertical and horizontal error bars. default: 0.
PlotNameA[M]	ch	1	Name of plot curve
PlotNameB[M]	ch	1	Name of plot curve, B channel (can be used to specify a second channel, e.g. B channel of transfer function, or B channel of cross-power spectrum).
PlotColumnA[M]	i	7	data columns: XY plot: X, Y, –, err1A, err2A, err1B, err2B; time series: –, Re Y, Im Y, err1A, err2A, err1B, err2B; histogram: X, Y, –, err1A, err2A, err1B, err2B; Bode plot: X, Re Y, Im Y, err1A, err2A, err1B, err2B. The error bar columns are used depending on the setting of ErrorBars. Default: 0, 1, 2, 3, 4, 5, 6.
PlotColumnB[M]	i	7	data columns; same as above, but for B channel.

Plooting packages which do not implement the complete set of above options have to implement as many as possible.

B.2.2 TIME SERIES

Time series containing raw channel information are stored by the channel name and two indices (measurement step and measurement point). Each of these data object contaoins the data of one channel only. The following table lists the parameters of a time series object:

Name	Type	Dim	Description
ObjectType	st	1	TimeSeries
Subtype	i	1	0 – normal time series in format (Y), 1 – averaged time series in format (Y), 2 – down-converted time series in format (Y), 3 – normal time series in format (t,Y), 4 – averaged time series in format (t,Y), 5 – down-converted time series in format (t,Y).
t0	ll	1	start time in GPS nsec.
dt	d	1	temporal spacing in sec.
f0	d	1	modulation frequency in Hz (type 2/5), trigger rate in Hz (type 1/4).
AverageType	i	1	0 – fixed number, 1 – running (exponential weight).
Averages	i	1	number of averages.
Channel	ch	1	channel name
N	ll	1	number of points.
Unit	st	1	physical unit.
	f / zf	N N×2	time series in format (Y), time series in format (t,Y).

Normally, raw data uses sub types 0 or 2, whereas a time series of a result object uses sub type 1.

B.2.3 FFT AND (CROSS) POWER SPECTRUM

A data object which describes FFT and power spectra can contain multiple spectra. By convention the first one always describes the averaged spectrum. The following table lists the parameters associated with a power spectrum:

Name	Type	Dim	Description
Type	st	1	Spectrum
Subtype	i	1	0 – FFT in format (Y), 1 – power spectral density in format (Y), 2 – cross-power spectrum in format (Y), 3 – FFT in format (f, Y), 4 – power spectral density in format (f,Y), 5 – cross-power spectrum in format (f,Y).
f0	d	1	start frequency in Hz.

Name	Type	Dim	Description
df	d	1	frequency spacing in Hz.
t0	ll	1	start time in GPS nsec.
dt	d	1	temporal spacing in sec (only useful for averaged power spectrum).
Window	i	1	0 – uniform (no window), 1 – Hanning window, 2 – Flat-top.
AverageType	i	1	0 – fixed number, 1 – running (exponential weight).
Averages	i	1	number of averages (only useful for power spectra).
ChannelA	ch	1	channel name
ChannelB	ch	1	2nd channel name for cross-power spectrum
N	ll	1	number of points.
M	ll	1	number of spectra; the first one is always the averaged spectrum. If individual spectra are stored, then M = Average + 1.
Unit	st	1	physical unit.
	f f / zf	N×M +N N×M +N	spectral density in format (Y), spectral density in format (f,Y), FFT spectrum in format (Y), FFT spectrum in format (f,Y).

B.2.4 TRANSFER FUNCTION AND COHERENCE

A transfer function object can contain multiple transfer functions of the same two measurement points. By convention the first one is the average of the following ones. The following list presents the associated parameters of a transfer function:

Name	Type	Dim	Description
ObjectType	st	1	TransferFunction
Subtype	i	1	0 – transfer function B/A in format (Y), 1 – transfer function A in format (Y), 2 – coherence B/A in format (Y), 3 – transfer function B/A in format (f,Y), 4 – transfer function A in format (f,Y), 5 – coherence B/A in format (f, Y).
f0	d	1	start frequency in Hz.
df	d	1	frequency spacing in Hz.
t0	ll	1	start time in GPS nsec.
BW	d	1	measurement bandwidth in Hz.

Name	Type	Dim	Description
AverageType	i	1	0 – fixed number, 1 – running (exponential weight).
Averages	i	1	number of averages.
ChannelA	ch	1	name of A channel
ChannelB	ch	1	name of B channel
N	ll	1	number of points.
M	ll	1	number of transfer functions; the first one is always the averaged transfer function. If individual transfer functions are stored, then M = Average + 1.
	zf	N×M +N	transfer function in format (Y), transfer function in format (f,Y).

B.2.5 LIST OF COEFFICIENTS

A sine response measurement can yield multiple transfer coefficients which are stored in two dimensional arrays. One of the dimension always represents the multiple measurement points, whereas the other dimension may represent the multiple excitation points, the harmonic order or the modulation product terms.

Name	Type	Dim	Description
ObjectType	st	1	Coefficients
Subtype	i	1	0 – transfer coefficients, 1 – harmonic coefficients, 2 – intermodulation product, 3 – coherence coeffcients
t0	ll	1	start time in GPS nsec.
f0	d	1	frequency in Hz.
f1	d	1	second frequency in Hz for intermodulation products.
BW	d	1	measurement bandwidth in Hz.
AverageType	i	1	0 – fixed number, 1 – running (exponential weight).
Averages	i	1	number of averages.
Channel[M]	ch	1	channel name corresponding to M-th detection point.
N	ll	1	number of frequency points.
M	ll	1	number of detection points.
Unit[M]	st	1	physical unit.
	zf	M×N M×N M×4	transfer coefficients: M × (f1, f2, f3, ...), harmonic coefficients: M × (f1, 2 f1, 3 f1, ...), intermodulation product: M × (f1, f2, |f1–f2|, f1+f2).

B.2.6 Measurement Values

Measurement values which do not fit into one of the above category can be stored in a table:

Name	Type	Dim	Description
ObjectType	st	1	MeasurementTable
t0	ll	1	start time in GPS nsec.
TableLength	i	1	length of table
Name[N]	st	1	name of measurement variable
Unit[M]	st	1	physical unit.
Description[M]	st	1	description of measurement
ValueType[M]	st	1	type of value: number or complex
	zf	M	measurement values

B.3 Diagnostics Tests

B.3.1 Sine Response, Harmonic Distortion and Two-Tone Intermodulation Tests

Name	Type	Dim	Description
ObjectType	st	1	TestParameter
Subtype	st	1	SineResponse
MeasurementTime	d	2	measurement time at each frequency; first value in sec, second value in cycles. The smaller one is taken and rounded up to the next cycle. Negative values are ignored. default: 100, 10.
Averages	i	1	number of averages; default is 1.
SettlingTime	d	2	settling time at each frequency step; first value in sec, second value in cycles. The smaller one is taken. Negative values are ignored. default: 10, 3.
StimulusChannel[M]	ch	1	name of stimulus channel.
StimulusFrequency[M]	d	1	frequency of stimulus channel in Hz. default: 100.
StimulusAmplitude[M]	d	1	amplitude of stimulus channel. default: 1.
StimulusOffset[M]	d	1	offset of stimulus channel. default: 0.

Name	Type	Dim	Description
StimulusPhase[M]	d	1	phase of stimulus signal in rad; the phase is relative to the last 0:00UTC. default: 0.
StimulusWait[M]	d	1	settling time of stimulus in sec. default is 0.25sec.
MeasurementChannel[N]	ch	1	measurement channel.
FFTResult	b	1	if true (default), calculates averaged 1024 point FFTs of each measurement channel as part of the result.

If there is only one stimulus channel, the result will automatically include a harmonic analysis of the first stimulus channel at every measurement point. Similarly, if there are exactly two stimulus channels of different frequencies, the result will include a two-tone intermodulation analysis at every measurement point. In all cases the result will include a sine reponse analysis of every stimulus frequency at every measurement point.

B.3.2 SWEPT SINE TESTS

Name	Type	Dim	Description
ObjectType	st	1	TestParameter
Subtype	st	1	SweptSine
SweepType	i	1	0 – linear, 1 – log, 2 – user supplied. default: 1.
SweepDirection	i	1	0 – upwards, 1 – downwards (default).
StartFrequency	d	1	start frequency of swept sine in Hz. default: 1.
StopFrequency	d	1	stop frequency of swept sine in Hz. default: 1000.
NumberOfPoints	i	1	number of frequency steps; default is 61.
FrequencySteps	d	M	user supplied frequency steps.
Averages	i	1	number of averages; default is 1.
MeasurementTime	d	2	measurement time at each frequency; first value in sec, second value in cycles. The smaller one is taken and rounded up to the next cycle. Negative values are ignored. default: 100, 10.

Name	Type	Dim	Description
SettlingTime	d	2	settling time at each frequency step; first value in sec, second value in cycles. The smaller one is taken. Negative values are ignored. default: 10, 3.
StimulusChannel	ch	1	name of stimulus channel.
StimulusAmplitude	d	1	amplitude of stimulus channel. default: 0.
MeasurementChannel[N]	ch	1	measurement channels; must be at least two. By default the first channel is the A channel.
FFTResult	b	1	if true, calculates averaged 1024 point FFTs of each measurement channel for each frequency step as part of the result; default is false.

B.3.3 FOURIER TESTS

Name	Type	Dim	Description
ObjectType	st	1	TestParameter
Subtype	st	1	FFT
StartFrequency	d	1	start frequency of FFT in Hz. default: 0.
StopFrequency	d	1	stop frequency of FFT in Hz; internally always rounded up so that the frequency spawn is a power of 2. default: 1000.
BW	d	1	bandwidth in Hz; always rounded to the closest power of 2. default: 1.
Overlap	d	1	overlap of FFT windows (only useful when averaging); t(i) = t0 + i * (1 – Overlap) * dt. default: 0.5.
Window	i	1	0 – uniform (no window), 1 – Hanning window, 2 – Flat-top.
AverageType	i	1	0 – fixed number, 1 – running (exponential weight). default: 0.
Averages	i	1	number of averages; default is 10.
MeasurementChannel[N]	ch	1	measurement channel.

B.3.4 Time Series Measurements and Trigger Response Tests

Name	Type	Dim	Description
ObjectType	st	1	TestParameter
Subtype	st	1	TimeSeries
TriggerRate	d	1	trigger rate in sec; if no trigger is used, this is the measurement duration.
PreTriggerTime	d	1	measurement time before trigger is applied; default is 20% of the total measurement time.
TriggerType	i	1	0 – no trigger signal, 1 – square wave, 2 – impulse, 3 – ramp, 4 – triangle. default: 0.
TriggerNum	i	1	number of triggeres/averages. 1 – single trigger response (no average), >1 – periodic trigger response (with average), default: 10.
AverageType	i	1	0 – fixed number, 1 – running (exponential weight).
TriggerChannel	ch	1	name of trigger channel
TriggerAmplitude	d	1	amplitude of trigger signal
MeasurementChannel[N]	ch	1	measurement channel.

B.3.5 Random Stimulus Response Tests

TBD.

B.4 XML Conventions

The file format for saving a diagnostics test is the LIGO lightweight file format which is based on XML. Data fields of parameters are ASCII encoded; in case the parameter uses a list of values they are comma delimited. Data fields of data objects are save as binary arrays following the C convention for storing rows and columns and using a big-endian representation. These binary data fields are appended to the XML file, so that a diagnostics test can be stored in a single file.

APPENDIX C SOFTWARE MODULES

A fair amount of documentation is written directly into the header files of the C and C++ modules. This documentation can be converted into html web pages using the doc++ program. The web pages can be reached at www.ligo-wa.caltech.edu/gds. This appendix is intended to give an overview of the existing software modules and to explain their main functions. A detailed description of the constants, types, macros, objects, routines and their parameters is found in the web pages.

C.1 OVERALL STRUCTURE

The source tree of the diagnostics test software is divided into sections (sub directories) representing groups of modules. A typical section contains all the modules associated with a software interface or category. The following sections are implemented:

Section	Description	Modules
src/algorithm	analysis algorithms: FFT, swept sine, etc.	decimate – decimation gdsrand – random number generation gdssigproc – general purpose signal processing sineanalyze – sine amplitude determination
src/awg	arbitrary waveform generator, excitation engine	awg – excitation generator and manager awgapi – API to the excitation engine awgfunc – general purposes awg functions awgtype – types used by the excitation engine awg_server – rpc server of excitation engine excitation – diagnostics test interface to the awg rawgapi – rpc (remote procedure call) interface
src/cmd	command line interface	cmdline – command line interpreter gdscmd – cmd line interface to diagnostics kernel gdsmsg – message interface for sending commands gdsmsg_server – rpc server for command messages rgdsmsg – rpc interface
src/daq	interface to the data acquisition system	gdschannel – interface to channel information database gdsrtdd – interface to the network data server
src/diag	diagnostics tests	diagclass – basic class for diagnostics tests diagnames – names used by tests repeat – repeats a test sineresponse – sine response test sweptsine – swept sine test testiter – basic object for test iterations testorg – manages all diagnostics tests

Section	Description	Modules
src/drv	hardware drivers	cobox – driver for an ethernet-to-RS232 converter ds340 – driver for SR DS340 signal generators gdsdac – driver for ICS115 digital-to-analog converter gpsclk – driver for the VME gps clocks hardware – parameters of the VME modules rmapi – driver for reflective memory boards target – VME host parameters
src/prog	programs	chndump – dumps the channel database to screen chnsave – saves channel data to disk diag – main program gdsd – diagnostics kernel (daemon)
src/rmem	reflective memory interface	map – memory map of the reflective memory rmorg – macros for managing DCUs testpoint – API for selecting test points testpoint_server – test point manager rtestpoint – rpc interface
src/sched	scheduler	gdssched – scheduler gdssched_client – remote scheduler client interface gdssched_server – remote scheduler server interface gdsrsched – rpc interface
src/storage	storage objects for diagnostics tests	gdsdatum – hierarchical storage object diagdatum – diagnostics storage object rtddinput – storage interface to the network data server
src/test	test programs	miscellaneous
src/util	utilities	gdserr – common error log gdserrmsg – error messages gdsheartbeat – heartbeat interface and synchronization gdsmain – compiler directives for different hosts gdsmutex – mutual exclusion semaphore objects gdsprm – parameter file interface gdsstring – additional string functions gdstask – task/thread creation rpcinc – common rpc routines tconv – time conversion routines gdsutil – includes most utility modules

C.2 UTILITIES

The following sections list the main routines and objects of the more important software modules.

C.2.1 GDSERR

Function	Description
gdsConsoleMessage	prints a message to the gds console
gdsErrorMessage	prints an error message
gdsError	prints one of the predefined error conditions
gdsWarningMessage	prints a warning message
gdsDebugMessage	prints a debug message
gdsDebug	prints a debug message if the DEBUG is defined

C.2.2 GDSHEARTBEAT

Function	Description
installHeartbeat	installs a heartbeat interrupt (16 Hz clock derived from GPS)
syncWithHeartbeat syncWithHeartbeatEx	synchronizes the program execution to the next heartbeat

C.2.3 GDSMUTEX

Object	Description
mutex	mutex object
recursivemutex	mutex which can be called multiple times from the same thread
readwritelock	read/write lock semaphore
semlock	locks a semaphore/mutex as long during the scope of the object

C.2.4 GDSPRM

Function	Description
findParamFileSection	finds a section within a parameter file
nextParamFileSection	finds the next section
getParamFileSection	gets a section
findParamSectionEntry	finds a parameter section entry
nextParamSectionEntry	finds the next entry
getParamSectionEntry	gets the entry

Function	Description
loadParamSectionEntry	loads a parameter
loadBoolParam	loads a boolean parameter
loadIntParam	loads an integer parameter (int)
loadNumParam	loads a numerical value (unsigned long)
loadFloatParam	loads a floating point number (double)
loadStringParam	loads a string parameter

C.2.5 GDSSTRING

Function	Description
gds_strcasecmp	compares two strings ignoring the case
strend	goes to the end of a string
strecpy	copies a string and returns the end of the resulting string
chnIsValid	returns true if a correctly formatted channel name is supplied
chn...	miscellaneous channel name handling functions

C.2.6 GDSTASK

Function	Description
taskCreate	creates a new task/thread

C.2.7 RPCINC

Function	Description
rpcGetHostaddress	returns the host address
rpcGetLocalAddress	gets the address of the local machine
rpcGetClientAddress	gets the rpc client address
rpcInitializeServer	initializes an rpc server
rpcRegisterService	registers an rpc service
rpcStartServer	starts an rpc server
rpcStartCallbackService	starts an rpc callback service
rpcStopCallbackService	terminates the rpc callback service
rpcResgisterCallback	registers a callback service
rpcProbe	test if an rpc service exists

C.2.8 TCONV

Function	Description
TAInow	returns the time in GPS seconds
TAItoUTC UTCtoTAI	converts between universal coordinate time and gps seconds

C.3 ALGORITHMS

C.3.1 DECIMATE

Function	Description
decimate	filter decimation stage
zoom	down-conversion

C.3.2 GDSRAND

Function	Description
urand_r	MT safe uniformly distributed random number generator
urandv_r	uniformly distributed random vector generator
nrand_r	normally distributed random number generator
nrandv_r	normally distributed random vector generator

C.3.3 GDSSIGPROC

Function	Description
DotProd	vector dot product
Mean	vector mean
Mixdown	complex down-conversion

C.3.4 SINEANALYZE

Function	Description
sineAnalyze	determines the amplitude and phase of a sine wave
sweptSineNpts	calculates the number of necessary data points

C.4 ARBITRARY WAVEFORM GENERATOR

C.4.1 AWG

Function	Description
initAwg	initializes the arbitrary waveform generator
getIndexAWG	gets an unused slot in the awg
releaseIndexAWG	frees an awg slot
resetAWG	resets an awg slot
configAWG	configures an awg from a parameter file
showAWG	displays the current state information of an awg slot
processAWG	calculates the waveform
disableAWG	disables a slot
enableAWG	enables a slot
addWaveformAWG	adds a waveforms to a slot
setWaveformAWG	sets an rabbitry waveform vector
queryWaveformAWG	returns the current waveforms
checkConfigAWG	checks the configuration
resetAllAWG, configAllAWG, showAllAWG, processAllAWG, disableAllAWG, enableAllAWG	same as corresponding functions above but for all awg slots
getStatisticsAWG	get the real-time performance statistics of an awg

C.4.2 AWGAPI

Function	Description
awg_client	installs the awg client interface
awgSetChannel	reserves a slot of an awg
awgRemoveChannel	frees the slot
awgAddWaveform	adds waveforms (signals)
awgSetWaveform	sets an arbitrary waveform vector
awgQueryWaveform	returns the current waveforms
awgReset	resets an awg

Function	Description
awgStatistics	returns the real-time performance statistics of an awg
awgShow	shows the status information of an awg
awgCommand	command interpreter for arbitrary waveform generators

C.4.3 AWGFUNC

Function	Description
normPhase	normalizes the phase to be between 0 and 2π
productLog	calculates the inverse of z(w)=wexp(w)
awgSignal	calculates a waveform function
awgPhaseIn	phase-in function
awgPhaseOut	phase-out function
awgSweepOut	phase-out for frequency sweep
awgSweepComponents	defines a frequency sweep waveform component
awgPeriodicComponent	defines a waveform component of a periodic waveform
awgIsValidComponent	tests if waveform component is valid
awgSortComponents	sorts awg components according to their start time

C.4.4 AWG_SERVER

Function	Description
awg_server	starts the rpc services for an arbitrary waveform generator

C.4.5 EXCITATION

TBD.

C.5 COMMAND LINE INTERFACE

C.5.1 CMDLINE

Object / Method	Description
commandline	command line object
operator !	returns true if finished
operator ()	calls the command line interpreter

C.5.2 GDSCMD

Function	Description
gdsCmdInit	Initializes the diagnostics kernel
gdsCmdFini	terminates the diagnostics kernel
gdsCmd	executes a diagnostics command
gdsCmdNotifyHandler	installs a command notification handler
cmdNotification	sends a notification back to the command line interface

C.5.3 GDSMSG

Function	Description
gdsMsgOpen	opens a communication channel to the diagnostics kernel
gdsMsgClose	closes the communication channel
gdsMsgSend	sends a message to the diagnostics kernel
gdsMsgInstallHandler	installs a message callback handler for receiving notifications

C.5.4 GDSMSG_SERVER

Function	Description
gdsmsg_server	starts the rpc message server of the diagnostics kernel

C.5.5 GDSMSG_SOCKETS

Function	Description
gdsmsg_sockets	starts the TCP/IP message server of the diagnostics kernel

C.6 DATA ACQUISITION INTERFACE

C.6.1 GDSCHANNEL

Function	Description
gdsChannelInfo	obtains channel information
gdsChannelListLen	returns the number of channels in the database
gdsChannelList	returns a list of all channels

C.6.2 GDSRTDD

Function	Description
gdsSubscribeData	subscribes a channel to the network data server
gdsUnsubscribeData	unsubscribes the channel
gdsGetNewdata	gets new channel data
gdsGetData	gets channel data for a specific time interval

C.7 DIAGNOSTICS TESTS

TBD.

C.8 HARDWARE DRIVERS

The layout of the excitation engine VME modules can be found in the hardware and target header files.

C.8.1 COBOX

Function	Description
openCobox	opens a socket connection to a cobox (ethernet-to-RS232 converter)

C.8.2 DS340

Function	Description
connectSerialDS340	connects a DS340 through a serial port
connectCoboxDS340	connects a DS340 through a cobox
resetDS340	resets a DS340
isDS340Alive	tests if the DS340 is alive
pingDS340	tests if the DS340 is connected and powered up
setDS340 uploadDS340Block	sets the configuration of a DS340
getDS340 downDS340Block	gets the configuration
sendWaveDS340	uploads an arbitrary waveform vector to a DS340
sendResetDS340	sends a reset signal
sendClearDS340	sends a clear signal
sendTriggerDS340	sends a trigger signal

C.8.3 GDSDAC

Function	Description
dacInit	initializes the ICS115 board
dacRestart	restarts the ICS115 and re-synchronizes it with the GPS clock
dacCopyData	copies data to the ICS115

Not yet implemented.

C.8.4 GPSCLK

Function	Description
gpsBaseAddress	returns the VME base address of the GPS clock
gpsInit	initializes the board
gpsSyncInfo	returns the synchronization status
gpsTimeNow	returns the current time in GPS nsec
gpsTime	converts the native time format of the board into GPS nsec
gpsNativeTime	returns the current time in the native format
gpsMicroSec	returns the micro seconds only
gpsInfo	returns GPS information
gpsHeartbeatInstall	Installs a 16 Hz GPS synchronized interrupt
gpsHeartbeatHealth	monitors the health of the heartbeat interrupt

C.8.5 RMAPI

Function	Description
rmInit	initializes the reflective memory board
rmBaseAddress	returns the base address of the reflective memory region
rmBoardAddress	returns the base address of the reflective memory board
rmBoardSize	returns the memory size supported by the board
rmLED	turns the LED on and off
rmResetNode	resets a reflective memory node
rmInt	sends an interrupt to a reflective memory node
rmCheck	check if a memory region is accessible
rmRead	reads from reflective memory
rmWrite	writes to reflective memory

C.9 Reflective Memory Organization

The memory map of the reflective memory and set of macros to handle parameters of data collection units can be found in the header files of the map and rmorg modules.

C.9.1 TESTPOINT

Function	Description
testpoint_client	initializes the test point client interface
tpRequest	selects a set of test points
tpClear	clears test points
tpQuery	returns the active test points
tpGetIndexDirect	returns the test point index on VME systems
tpIsValid	tests if a test point is valid
tpAddr	returns the address of a test point in reflective memory
tpCommand	command line interface to the test point manager

C.9.2 TESTPOINT_SERVER

Function	Description
testpoint_server	starts the rpc services of the test point manager

C.10Scheduler

C.10.1GDSSCHED

Function	Description
createScheduler	creates a scheduler
closeScheduler	closes a scheduler
scheduleTask	adds a new task to the scheduler
getScheduledTask	obtains information about a scheduled task
removeScheduledTask	removes a task from a scheduler
waitForSchedulerToFinish	waits until all tasks have been scheduled and finished
setScheduleTag	sets a synchronization tag

C.10.2GDSSCHED_CLIENT

Function	Description
createRemoteScheduler	creates a new local scheduler for remote use
createBoundScheduler	creates a new scheduler on a remote machine

C.10.3GDSSCHED_SERVER

Function	Description
registerSchedulerClass	registers a task which can be scheduler from remote
runSchedulerService	start the rpc services of a remote scheduler manager

C.11STORAGE OBJECTS

The class hierarchy of the diagnostics storage interface is shown in Fig. 12. There are two main class categories: *(i)* classes which contain data (inherited from gdsDatum) and *(ii)* classes which manage the access to the data (inherited from diagObjectName). The main diagnostics storage object is of type diagStorage, it is inherited from a generic storage object, gdsStorage, and contains all access classes. This main object is able to set and get variables by name from the generic storage object using the access classes to make sure the variable names are valid test data object and valid test parameters. The storage object is organized implementing two hierarchical levels: the main storage objects contains both data objects and global parameter objects; whereas data objects can contain their own private parameter objects. Thus, the generic storage object, gdsStorage inherits from the data object, gdsDataObject, and also contains a list of data objects.

Access classes for data objects are inherited from diagObject which itself is inherited from diagObjectName. The diagObject contains a list of access classes for the parameters which are associated with this data object. Data objects which contain results and data objects which describe a diagnostics test are inherited from diagMultipleObject. The diagMultipleObject manages a list of access classes which are used depending on the type of the data object.

C.11.1GDSDATUM

Object / Method	Description
gdsDatum	object which stores data, can be multi-dimensional.
gdsNamedStorage	object with a name.
gdsNamedDatum	object which stores data and has a name.
gdsDataReference	object which manages data references to memory mapped files.
storage_ptr	auto pointer object which manages pointers to parameter and data objects.

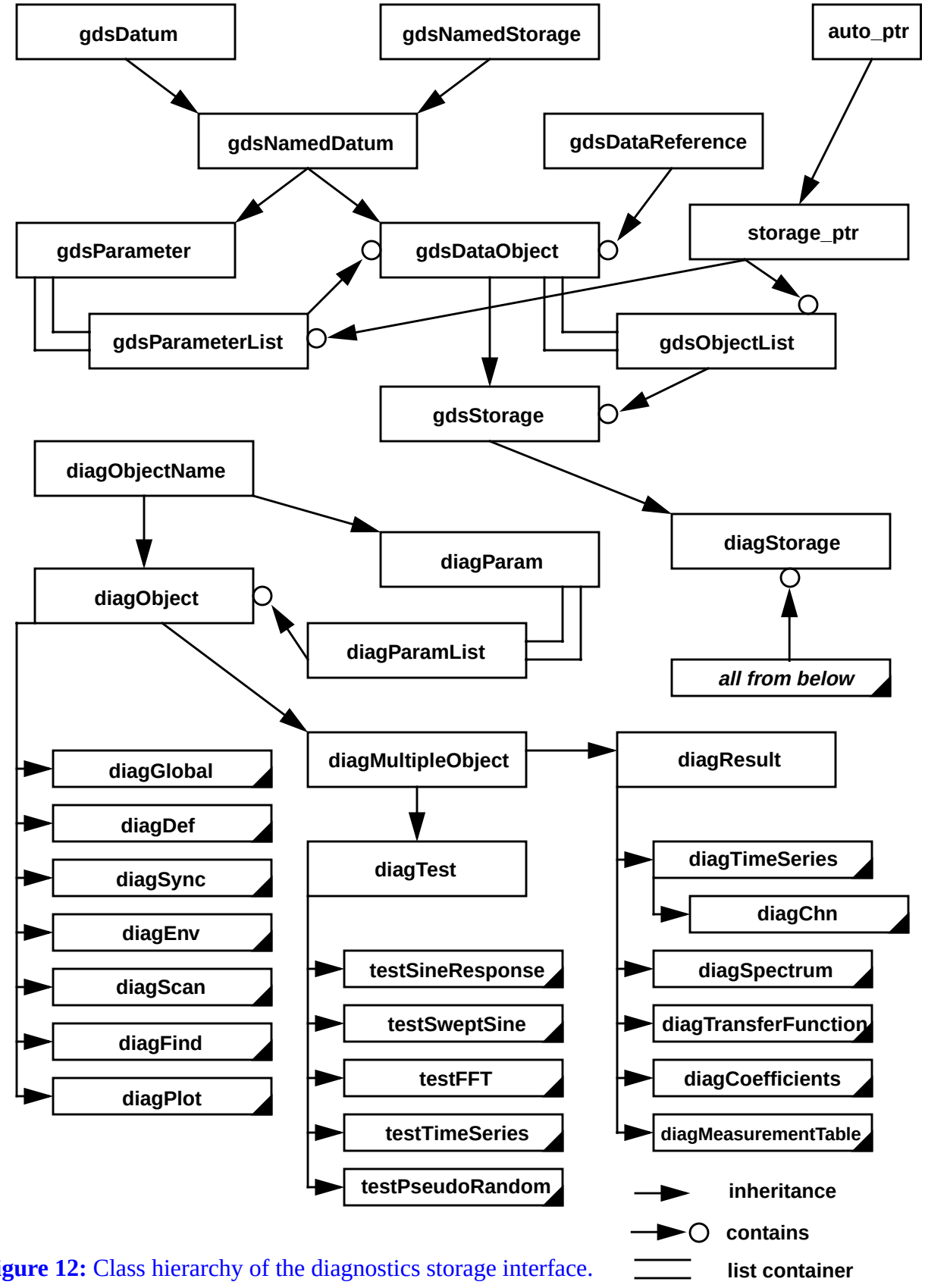


Figure 12: Class hierarchy of the diagnostics storage interface.

Object / Method	Description
gdsParameter	parameter object.
gdsDataObject	data object; contains a list of parameter objects.
gdsStorage	generic storage object; contains a list of data objects.

C.11.2DIAGDATUM

Object / Method	Description
diagObjectName	generic access class, handles names, dimensions, indices, data types and access writes.
diagParam	generic access class for a parameter object.
diagObject	generic access class for data objects.
diagGlobal, diagDef, diagSync, diagEnv, diagScan, diagFind, diagPlot	access class for global parameters, default parameters, synchronization variables, environment settings, scan parameters, optimization parameters and plot settings, respectively.
diagMultipleObject	generic access class for a data object with multiple possible configurations.
diagTest	access class for the diagnostic test object.
testSineResponse, testSweptSine, testFFT, testTimeSeries, testPseudoRandom	access class for sine response test, swept sine test, FFT test, time series measurement, and pseudo random response test, respectively.
diagResult	access class for a result object.
diagTimeSeries, diagChn diagSpectrum, diagCoefficients, diagMeasurementTable	access class for time series vectors, channel data, FFT spectra, measurement coefficients, and measurement tables, respectively.
diagStorage	diagnostics storage object; manages parameters and data objects which can have their own parameters. Uses the access classes to make sure that only valid parameter and data objects are stored.

C.11.3RTDDINPUT

Object / Method	Description
rtddcallback	class implementing a callback method for the network data server API
gdsRTDDchannel	class for reading channel data from the network data server, it manages a single channel, knows how to handle data partitions and knows where to store the read data in the diagnostics storage object.
partition	class which describes a channel data partition
gdsRTDDinput	class which manages a list of gdsRTDDchannel objects, i.e. class which handles the input from the network data server.

C.12 PROGRAMS

C.12.1 CHNDUMP

This is a utility program to dump all channel information records to the standard output.

C.12.2 DIAG

This program is the interface to the diagnostic system, it can invoke either the command line interface or the graphical user interface. For a more detailed discussion see Sections 2.2 – 2.4.

C.12.3 GDSD

This is the diagnostics kernel. See Appendix A on how to set it up.

C.12.4 LIBGDS.SO

This is the dynamic link library which implements most of the diagnostics kernel. It is used by the ‘diag’ program if a connection to a local kernel is established.