

LDAS Data Conditioning API: Mock Data Challenge #1

Lee Samuel Finn, James Kent Blackburn,
Philip Charlton, Phil Ehrens, Albert Lazzarini,
Ed Maros, Joseph Romano, Antony Searle
LIGO-T000124-00-E

Target date: 31 July – 7 August 2000

Abstract

First Mock Data Challenge and system integration test plan for the LDAS Data Conditioning API. Tests the basic functionality of the LDAS Data Conditioning API, focusing on correctness, through-put, capacity (i.e., number of independent, parallel analyses), and interoperability of the components

1 Test Definition

1.1 Overview

The purpose of this MDC is to verify or quantify the functionality of the LDAS Data Conditioning API, focusing on

- Correctness of implementation;
- Integration of programming elements (C++ & Tcl);
- Integration of system elements (Data Conditioning, Assistant Manager, and Manager APIs);
- Ability to handle parallel, concurrent analyses;
- Through-put and capacity.

1.2 Hardware and Software Components to be tested

1.2.1 Software components to be tested

- Correctness of implementation: A subset of the functionality described in the Data Conditioning API baseline (LIGO T990002-00-E), consisting of quadrature amplitude demodulation, generation of summary statistics, power spectral density estimation, a DFT/IDFT pair, FIR/IIR linear filtering, decimation and upsampling.
- Integration of programming elements: the functionality described above and the Tcl management of the DataConditioningAPI;

- Integration of system elements: the ability to drive the data conditioning API from user commands submitted to the ManagerAPI;
- Ability to handle parallel, concurrent analyses: the ability of the data conditioning API to successfully dispatch and manage multiple ldas jobs in separate threads;
- Throughput and capacity: the baseline subset described above as driven by user commands submitted to the ManagerAPI.

1.2.2 Hardware components to be tested

None.

1.3 Output data to be generated

In addition to the completed test checklists (see appendix B), this MDC produces a set of performance statistics, describing the performance of the MDC objects at the C++ level and Tcl command language level.

2 Test Results

This section summarizes the results produced by this MDC.

2.1 Test Checklists

Each test that forms a part of this MDC is fully described in a test checklist (see appendix B). Each step in any particular test is described on the appropriate checklist. The successful or unsuccessful completion of each step is recorded on the checklist together with any responses. These checklists are fully completed and part of the official record of this MDC.

2.2 Performance Statistics

The Data Conditioning API interprets a complex user command, which may consist of a sequence of discrete actions, and dispatches the entire command sequence into a single thread. As part of this MDC we determine the typical user time within the thread required to perform certain critical data conditioning steps within the thread.

3 Criteria for evaluation

3.1 Software

Correctness of results: TBD

Error handling: TBD

Throughput: 72h continuous operation at full throughput with no evidence of memory leaks

Capacity: meet specification with TBD safety margin

Interoperability: TBD

3.2 Hardware

No hardware is tested in this MDC.

4 Execution Plan

Prior to beginning this MDC assemble all the required components (cf. 4.1). The version of LDAS and the data conditioning API that are being tested are tagged in the repository with PREMDCYYMMDD, where YY is the current year, MM the two-digit current month and DD the current day. This version of LDAS and the data conditioning API is referred to here as the test-version.

4.1 Required Input Data, Hardware, Software, Personnel

4.1.1 Required input data

- API Tests: [[NAME INPUT FILES]]
- Pipeline Tests: [[NAME INPUT FILES]]
- Long-term operations test: [[NAME INPUT FILES]]

4.1.2 Required Hardware

Data Conditioning Unit: this MDC is run on the hardware that will support LDAS data conditioning.

4.1.3 Required Software

TBD LDAS Components

4.1.4 Required Personnel

- Data Conditioning API development team
- LDAS Development Team

4.2 Environment

This MDC runs on the data conditioning hardware. It assumes a non-superuser account on that system, which is referred to here and henceforth as `mdc`. At the beginning of the MDC the `mdc` account is assumed to have an appropriate path, read access to the LDAS S/W Repository, and adequate disk space to download and build LDAS, including the data conditioning API.

4.3 Documentation

Verify that each components documentation is complete: Worksheets [[WORKSHEET REFERENCES]]

4.4 Build Test

The API and its tests should build successfully within LDAS with no errors or warnings: Worksheets [[WORKSHEET REFERENCES]]

4.5 Library Level Tests

The API components should pass their tests programs with no errors or warnings: Worksheets [[WORKSHEET REFERENCES]]

4.6 API Level Tests

Exception handling and correctness testing of the functionality exposed through the API.

4.6.1 Mixer

- Exception testing: Test that `std::domain_error` and `std::invalid_argument` exceptions thrown by mixer (corresponding to out-of-range frequency and zero-length input sequence) are caught and recorded at the Tcl layer.
- Evaluation testing: Test that the mixer `apply` method returns correct results when acting on a valid input sequence, as a whole or when the sequence is broken into two parts.

4.6.2 Statistics

- Exception testing: Test that `std::invalid_argument` exceptions thrown by the `min`, `max`, `mean`, `rms`, `variance`, `skewness`, `kurtosis`, and `all` methods (when acting on a zero-length sequence) are caught and recorded at the Tcl layer.
- Evaluation testing: Test that correct results are returned by the `size`, `min`, `max`, `mean`, `rms`, `variance`, `skewness`, `kurtosis`, and `all` methods when acting on a valid input sequence.

4.6.3 Linear Filtering

- Exception testing: Test that `std::invalid_argument` exceptions thrown by linear filter (corresponding to zero-length FIR filter coefficients, zero-length IIR filter coefficients, and zero-length input sequence) are caught and recorded at the Tcl layer.
- Evaluation testing: Test that the linear filter `apply` method returns correct results when acting on a valid input sequence, as a whole or when the sequence is broken into two parts.

4.6.4 Resampling

- Exception testing: Test that exceptions thrown by `resample` (corresponding to $p < 1$, $q < 1$, both $p, q > 1$, input sequence length not a multiple of q , and invalid state length) are caught and recorded at the Tcl layer.

- Evaluation testing: Test that the resample `apply` method returns correct results when acting on a valid input sequence, as a whole or when the sequence is broken into two parts. Unbroken sequences (upsample by 5 and downsample by 5) and broken sequences (downsample by 5 and upsample by 2) are tested.

4.6.5 FFT/IFFT

- Exception testing: Test that `std::invalid_argument` exceptions thrown by the FFT method (when acting on sequences with invalid lengths: $N = 0$ and $N = \text{MaximumFFTLength} + 1 = 8388609$) are caught and recorded at the Tcl layer.
- Evaluation testing: Test that the FFT and IFFT `apply` methods return correct results for a range of valid input lengths ($N = 1, 1024, 122880,$ and 1843200) and that Parseval's theorem holds for an FFT of a sequence of length $N = \text{MaximumFFTLength} = 8388608$

4.6.6 Power Spectrum Estimation

- Exception testing: Test that `std::invalid_argument` exceptions thrown by Welch estimate (corresponding to zero length sequence, `fft length = 0`, `overlap length = fft length`, and `overlap length > fft length`) are caught and recorded at the Tcl layer.
- Evaluation testing: Test that correct results are returned by the Welch estimate `apply` method when acting on a valid input sequence of length 1024. The default estimate (`fft length = 1024`, `overlap length = 0`), and two custom estimates (`fft length = 256`, `overlap length = 0`; and `fft length = 256`, `overlap length = 128`) are tested.

4.7 Benchmarking

4.7.1 Mixer

- Figure or table showing user time to mix a sequence of N samples. The sequence length N will be in powers of 2 starting from 2^{10} samples and increasing to 2^{23} samples (equivalent to approximately 1h at a 1024 Hz bandwidth). Separate performance statistics will be calculated for real float, real double, complex float, and complex double precision data types. Times will be measured using the data conditioning API's `user_time` action, which returns the process user time.

4.7.2 Linear Filtering

- Figures or tables showing user time to linear filter a sequence of N samples. The sequence length N will be in powers of 2 starting from 2^{10} samples and increasing to 2^{23} samples (equivalent to 1h at a 1024 Hz bandwidth). Separate performance statistics will be calculated for real float, real double, complex float, and complex double precision data types. Times will be measured using the data conditioning API's `user_time` action, which returns the process user time. Separate performance statistics will be measured for:
 - an order 12 FIR filter,
 - an order 12 IIR filter,
 - an order 12 pole-zero filter.

- Figures or tables showing user time to linear filter a sequence of 2^{20} samples as a function of FIR filter order. The filter order will be in multiples of five beginning with order 5 and extending to order 50. Separate performance statistics will be calculated for real float, real double, complex float, and complex double precision data types. Times will be measured using the data conditioning API's `user_time` action, which returns the process user time.

4.7.3 Resampling

- Figures or tables showing user time to downsample a sequence of N samples. The sequence length N will be in powers of 2 starting from 2^{10} samples and ranging to 2^{23} samples (equivalent to 512 s at 16 KHz bandwidth). Separate performance statistics will be calculated for real float, real double, complex float, and complex double precision data types. Times will be measured using the data conditioning API's `user_time` action, which returns the process user time. Separate performance statistics will be accumulated for downsampling by a factor of 2 and by factor of 2^3 using the default parameters specified by `Resample`.
- Figures or tables showing user time to upsample a sequence of N samples. The sequence length N will be in powers of 2 starting from 2^{10} samples and ranging to 2^{23} samples (equivalent to 512 s at 16 KHz bandwidth). Separate performance statistics will be calculated for real float, real double, complex float, and complex double precision data types. Times will be measured using the data conditioning API's `user_time` action, which returns the process user time. Separate performance statistics will be accumulated for upsampling by a factor of 2 and by factor of 2^3 using the default parameters specified by `Resample`.

4.7.4 FFT/IFFT

- Figures or tables showing user time to FFT a sequence of N samples. The sequence length N will be all numbers of the form $2^l 3^m 5^n$, starting at 2^{10} and going up to and including the smallest power of 2 exceeding 2048×3600 (ie. 60 minutes of data sampled at 2048 Hz). Thus the maximum N will be $N = 2^{23} = 8388608$. Separate performance statistics will be calculated for real float, real double, complex float, and complex double precision data types. Times will be measured using the data conditioning API's `user_time` action, which returns the process user time.

4.8 Pipeline Testing

A “pipeline” is a chain of operations assembled at the Tcl layer. The chain moves data through a series of API-level operations. Several pipelines will be tested, involving different API level components

- *Octave analysis pipeline.* This pipeline reads data in segments of length [[TBD]]. The data is understood to represent IFO data at a nominal sampling rate of 16384 Hz. Several octaves are selected, using `Mixer` and `Resample`, and summary statistics (including power spectral density estimates) are generated.
 1. A segment of input data is read and passed several `Mixer` objects operating at 1024Hz, 512Hz, 256Hz, and 128Hz.
 2. Each `Mixer` feeds into a `Resample` object, which reduces the bandwidth to 512Hz, 256Hz, 128Hz, and 64Hz, respectively.

3. The result of each `Resample` object feeds into a `SummaryStatistics` object to produce one second trend summary statistics. These summary statistics leave the data conditioning unit as ILWD objects.
 4. The result of each `Resample` object is collected until 256s of data are available in each octave. This collection feeds into a `WelchEstimate` object to produce an estimated PSD in each channel. The frequency resolution will be 4 Hz for the 1024 Hz sample rate channel, 2 Hz for the 512 Hz channel, 1 Hz for the 256 Hz channel, and 0.5 Hz for the 128 Hz channel. The estimated PSDs will leave the data conditioning API as ILWD objects.
 5. This pipeline will run for 2048 seconds, generating 16 PSDs per channel and 2048 trend statistics per channel.
 6. The 16384 Hz input data will have a recognizable spectral character in each band in order that we can verify correctness of the pipelines ability to manipulate the data. Each band will have a distinct overall rms noise amplitude and include a calibration line, corresponding to a sinusoid of known frequency and amplitude. The position of the line in the band and its amplitude relative to the rms noise floor will be unique to each band.
- *Parallel Analysis.* The parallel analysis test verifies the ability of the API to execute parallel chains in threads.
 1. 8 identical chains are executed, in parallel, on 8 different input files.
 2. Each chain computes the DFT and then IDFT on 64s of data, which leave the data conditioning API as ILWD objects.
 3. This pipeline will execute for 32 minutes of data, producing 256 ILWD output files.
 4. The first and last generated output files will be checked for correctness.

4.9 Long-term operation

The long-term operations test will consist of the two pipelines identified in section 4.8 running “unattended” for 72h.

5 Conclusion

Following the *successful* completion of the MDC (all tests successfully passed, all performance statistics successfully accumulated) the CVS repository subdirectory `ldas/api/datacondAPI` containing just that code base which was tested will be marked with the symbolic tag `datacondAPIMDC1`.

A C++ helper classes/functionality exposed to the API level

To support the implementation of the API, several helper classes were developed. All of these are tested as components at the C++ layer. Several of these are also exposed to the Tcl layer, where it is necessary to construct or manipulate them as named variables to implement the API. The necessary functionality of these helper classes is tested first in the C++ component level tests and later during the API level tests; no separate tests of these components is required. The following subsections call-out these helper classes and indicate which API level tests verify their correct functioning.

A.1 `datacondAPI::Window`

`FIRLP`, `KFIRLP`, and `WelchEstimate` use data conditioning objects of class `Window`. `Window` is an abstract base class with several different implementations (currently `RectangularWindow`, `HannWindow`, `KaiserWindow`). API level construction and manipulation of `Window` objects is tested by: [[SAM ADD CHECKLIST REFERENCE]], and `WelchEstimate02`.

A.2 `datacondAPI::fvalarray`

A.3 `std::valarray`, `std::slice`, `std::slice_array`

`std::valarray` is the basic carrier for sequences or subsequences. It is constructed and data is read-out from it at the Tcl layer. Additionally `std::slice` is used to subset `valarray` objects as `std::slice_array` objects, all at the Tcl layer. Construction of `valarray` objects is tested whenever sequence data enters the data conditioning API (*e.g.*, [[CHECKLIST REFERENCE]]) and the ability to read `valarray` objects is tested whenever sequence data is returned by the API, (*e.g.*, [[CHECKLIST REFERENCE]]). `slice` and `slice_array` objects are tested [[CHECKLIST REFERENCE]].

B Test Checklists

The following pages contain detailed procedures to be followed for testing the functionality of the Data Conditioning API at the API level.

Draft

Test Case: FFT00 (SAMPLE ONLY)

Purpose: To test that a `std::invalid_argument` exception thrown by the FFT apply method when acting on a zero-length sequence is caught and recorded at the Tcl layer.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N200XX.sim

where XX is either R4, R8, C8, or C16. These data sets consist of sequences of random numbers.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. Issue the LDAS command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/FFT00XX.ilwd
    -returnformat ilwd
    -aliases { idata = chan_01:data }
    -algorithms {
      x = slice(idata,0,0,1);
      rx = real(x);
      n = size(rx);
      intermediate(,,n,{length of input data});
      y=fft(x);
    }
    -resultcomment {none}
    -resultname {none}
  }
}
```

Pass / Fail

3. Verify that a `std::invalid_argument` error appears in the appropriate log file.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: MDCPREP01

Purpose: Prepare the environment for the Data Conditioning API MDC.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

ENVIRONMENT AND PREREQUISITES

The data conditioning API, running on the system `datacon`. The manager API, running on a separate LDAS system. Other APIs, as needed, running on appropriate hardware. A separate system, not running any of the LDAS APIs, through which the test team will interact with LDAS, referred to as `console`.

PROCEDURE

1. On the system `console` create the following directory hierarchy, which will hold input files and result files (including expected result files) of this MDC:

```

mdc
mdc/expect
mdc/input
mdc/lib
mdc/output
mdc/output/api
mdc/output/perf
mdc/output/pipe
mdc/output/ops

```



Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: BUILD01

Purpose: Verify the datacondAPI build, as part of the LDAS build

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

ENVIRONMENT AND PREREQUISITES

This test is executed from the user account `mdc` on the data conditioning hardware `datacon`.

PROCEDURE

1. Inspect the log-files from the current build of the LDAS S/W system and insure that LDAS, including the data conditioning api, builds without errors. Correct the LDAS source if necessary until a successful build results.

Pass / Fail

2. Mark the current, successfully built version of the LDAS S/W using the CVS tag `PRE01MDCYYMMDD`, where YY is the two digit year, MM the two digit month and DD the two digit day of the beginning of this MDC.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: BUILD02
Purpose: Verify the existence of the Data Conditioning API web-based class documentation.
Tester: _____
Test machine: _____
Date (mm/dd/yy): ____/____/____ **Time:** _____

ENVIRONMENT AND PREREQUISITES

This test is executed from the user account `mdc` on the data conditioning hardware `datacon`. It depends on the successful completion of BUILD01.

PROCEDURE

1. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `Mixer` class. **Pass / Fail**
2. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `MixerState` class. **Pass / Fail**
3. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `Resample` class. **Pass / Fail**
4. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `LinFilt` class. **Pass / Fail**
5. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `FIRLP` class. **Pass / Fail**
6. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `KFIRLP` class. **Pass / Fail**
7. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `Statistics` class. **Pass / Fail**
8. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `PSDEstimate` class. **Pass / Fail**

9. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `WelchEstimate` class.
Pass / Fail
10. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `FFT` class.
Pass / Fail
11. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `IFFT` class.
Pass / Fail
12. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `FFTImpBase` class.
Pass / Fail
13. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `SFFTImplement` class.
Pass / Fail
14. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `SIFFTImplement` class.
Pass / Fail
15. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `DFFTImplement` class.
Pass / Fail
16. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `DIFFTImplement` class.
Pass / Fail
17. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `pIandb` class.
Pass / Fail
18. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `plankey` class.
Pass / Fail
19. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `planhandle` class.
Pass / Fail
20. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `dplanhandle` class.
Pass / Fail
21. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `fvalarray` class.
Pass / Fail

22. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `Window` class. **Pass / Fail**
23. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `WindowImp` class. **Pass / Fail**
24. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `WindowInfo` class. **Pass / Fail**
25. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `HannWindow` class. **Pass / Fail**
26. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `KaiserWindow` class. **Pass / Fail**
27. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `RectangularWindow` class. **Pass / Fail**
28. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `Warning` class. **Pass / Fail**
29. Using a web-browser verify that the Data Conditioning API perceps documentation is built and complete for the `unimplemented_error` class. **Pass / Fail**

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: BUILD03
Purpose: Verify the existence of the Data Conditioning API
tex-based class documentation.
Tester: _____
Test machine: _____
Date (mm/dd/yy): ____/____/____ **Time:** _____

ENVIRONMENT AND PREREQUISITES

This test is executed from the user account `mdc` on the data conditioning hardware `datacon`. It depends on the successful completion of BUILD01.

PROCEDURE

1. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `Mixer` class.
Pass / Fail
2. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `MixerState` class.
Pass / Fail
3. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `Resample` class.
Pass / Fail
4. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `LinFilt` class.
Pass / Fail
5. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `FIRLP` class.
Pass / Fail
6. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `KFIRLP` class.
Pass / Fail
7. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `Statistics` class.
Pass / Fail
8. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `PSDEstimate` class.
Pass / Fail

9. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `WelchEstimate` class.
Pass / Fail
10. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `FFT` class.
Pass / Fail
11. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `IFFT` class.
Pass / Fail
12. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `FFTImpBase` class.
Pass / Fail
13. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `SFFTImplement` class.
Pass / Fail
14. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `SIFFTImplement` class.
Pass / Fail
15. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `DFFTImplement` class.
Pass / Fail
16. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `DIFFTImplement` class.
Pass / Fail
17. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `pIandb` class.
Pass / Fail
18. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `pIankeY` class.
Pass / Fail
19. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `pIanhandle` class.
Pass / Fail
20. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `dplanhandle` class.
Pass / Fail
21. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `fvalarray` class.
Pass / Fail

22. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `Window` class.

Pass / Fail

23. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `WindowImp` class.

Pass / Fail

24. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `WindowInfo` class.

Pass / Fail

25. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `HannWindow` class.

Pass / Fail

26. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `KaiserWindow` class.

Pass / Fail

27. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `RectangularWindow` class.

Pass / Fail

28. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `Warning` class.

Pass / Fail

29. Using a dvi-file previewer verify that the Data Conditioning API tex-based documentation is built and complete for the `unimplemented_error` class.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

TEST RESULT

Pass / Fail

Test Case: LIB01

Purpose: Verify the correct functioning of the C++ code that is a part of the data conditioning API.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

ENVIRONMENT AND PREREQUISITES

This test is executed from the user account `mdc` on the data conditioning hardware `datacon`. It depends on successful completion of BUILD01.

PROCEDURE

1. On the LDAS build system, in an account with r/w-access to the LDAS build tree, set the environment variable `$TEST_VERBOSE_MODE` to `true`. **Pass / Fail**
2. In the `$LDASOBSJS/api/datacondAPI` issue the (tcsh) commands


```
% date > $HOME/LIB01.out
% make check >>& $HOME/mdc/lib/LIB01.out
```

Pass / Fail
3. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tDFFT`:
 - (a) Verify that the program exits with exit status 0. **Pass / Fail**
 - (b) Verify that each component test returns a PASS. **Pass / Fail**

Correct the test code and/or the component code as necessary. **Pass / Fail**
4. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tDfftimplement`:
 - (a) Verify that the program exits with exit status 0. **Pass / Fail**
 - (b) Verify that each component test returns a PASS. **Pass / Fail**

Correct the test code and/or the component code as necessary. **Pass / Fail**
5. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tDplanhandle`:

- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
6. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tDrealfft`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
7. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tFFTPerf`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
8. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tFFT_thread`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
9. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tFFTW`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
10. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tFIRLP`:

- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
11. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tKFIRLP`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
12. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tLinFilt`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
13. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tMixer`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary.as necessary. **Pass / Fail**
14. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tPSDEstimate`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
15. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tPlandb`:

- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
16. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tPlandb_thread`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
17. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tPlankey`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
18. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tResample`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
19. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tSfft`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
20. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tSrealfft`:

- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
21. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tStatistics`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
22. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tUnimplemented`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
23. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tWindow`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
24. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tWarning`:
- (a) Verify that the program exits with exit status 0. **Pass / Fail**
- (b) Verify that each component test returns a PASS. **Pass / Fail**
- Correct the test code and/or the component code as necessary. **Pass / Fail**
25. Inspect the file `$HOME/mdc/lib/LIB01.out` for the results of the test program `tfvalarray`:

(a) Verify that the program exits with exit status 0.

Pass / Fail

(b) Verify that each component test returns a PASS.

Pass / Fail

Correct the test code and/or the component code as necessary.

Pass / Fail

26. Copy the file \$HOME/mdc/lib/LIB01.out to the directory console:mdc/lib/LIB01.out.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

TEST RESULT

Pass / Fail

Draft

Test Case: MIXERAPI01

Purpose: To test that `std::domain_error` and `std::invalid_argument` errors thrown by the mixer class (corresponding to out-of-range frequency and zero-length input sequence) are caught and recorded at the Tcl layer.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N200XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.
2. (`domain_error` in constructor) Issue `ldas` command

R4/ R8/ C8/ C16

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERAPI01AXX.ilwd
  -returnformat ilwd
  -resultname {None}
  -resultcomment {None}
  -aliases {raw=chan_01:data}
  -algorithms {
    x = slice(raw,0,100,1);
    phi = value(0.0);
    f = value(2.0);
    z = mix(phi,f,x);
  }
}
```

Pass / Fail

3. Verify that a `std::domain_error`, indicating an out-of-range frequency, appears in the appropriate log file.

Pass / Fail

4. (`invalid_argument` in apply method) Issue `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/MIXERAPI01BXX.ilwd
    -returnformat ilwd
    -resultname {None}
    -resultcomment {None}
    -aliases {raw=chan_01:data}
    -algorithms {
      x = slice(raw,0,0,1);
      phi = value(0.0);
      f = value(0.125);
      z = mix(phi,f,x);
    }
  }
}
```

Pass / Fail

5. Verify that a `std::invalid_argument` error, indicating a zero-length input sequence, appears in the appropriate log file.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: MIXERAPI02

Purpose: To verify that the mixer action returns correct values when acting on unbroken and broken input sequences.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following data sets are required for this test:

1. Input data:

(a) N200XX.sim

where XX is either R4, R8, C8, or C16. These data sets consist of sequences of random numbers.

2. Expected results:

(a) MIXERAPI02AXX.exp

where XX is R4, R8, C8, or C16. These data sets contain the expected results for the mixer action on unbroken sequences of length $N = 100$.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. (unbroken sequence) Issue the following ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol {
      file:/mdc/input/N200XX.sim
      file:/mdc/input/MIXERAPI02AXX.exp
    }
    -inputformat ilwd
    -returnprotocol file:/MIXERAPI02AXX.ilwd
    -returnformat ilwd
    -resultname {mix}
    -resultcomment {Mixer result}
```

```
-aliases {raw=chan_01:data; exp=mixout:data }
-algorithms {
  x = slice(raw, 0, 100, 1);
  f = value(0.125);
  phi = value(0.1);

  rx = real(x);
  ix = imag(x);
  variance(rx);
  intermediate(,rvarX, {Variance of real input data});
  rms(rx);
  intermediate(,rrmsX, {RootMeanSquare of real input data});
  variance(ix);
  intermediate(,ivarX, {Variance of imag input data});
  rms(ix);
  intermediate(,irmsX, {RootMeanSquare of imag input data});

  z = mix(phi, f, x);

  rz = real(z);
  rexp = real(exp);
  rdif = sub(rz, rexp);
  min(rdif);
  intermediate(,rminD, {Minimum real deviation});
  max(rdif);
  intermediate(,rmaxD, {Maximum real deviation});
  variance(rdif);
  intermediate(,rvarD, {Variance of real deviation});
  rms(rexp);
  intermediate(,rrmsE, {RootMeanSquare of real expected result});

  iz = imag(z);
  iexp = imag(exp);
  idif = sub(iz, iexp);
  min(idif);
  intermediate(,iminD, {Minimum imag deviation});
  max(idif);
  intermediate(,imaxD, {Maximum imag deviation});
  variance(idif);
  intermediate(,ivarD, {Variance of imag deviation});
  rms(iexp);
  intermediate(,irmsE, {RootMeanSquare of imag expected result});

  value(z);
}
}
```

Pass / Fail

3. Verify that the minimum and maximum values of the real and imaginary parts of the differences between the calculated and expected results are zero to floating point (or double) precision, relative to the rms amplitude of the expected results.

Pass / Fail

4. Move the output file to mdc/output/api.

Pass / Fail

5. (broken sequence) Issue the following ldas command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
  -returnprotocol file:/MIXERAPI02BXX.ilwd
  -returnformat ilwd
  -resultname {mixerbroken}
  -resultcomment {Broken mixer result}
  -aliases { raw=chan_01:data }
  -algorithms {
    x0 = slice(raw,0,100,1);
    f = value(0.125);
    phi = value(0.1);

    rx0 = real(x0);
    ix0 = imag(x0);
    variance(rx0);
    intermediate(,rvarX, {Variance of real input data});
    rms(rx0);
    intermediate(,rrmsX, {RootMeanSquare of real input data});
    variance(ix0);
    intermediate(,ivarX, {Variance of imag input data});
    rms(ix0);
    intermediate(,irmsX, {RootMeanSquare of imag input data});

    z0 = mix(phi,f,x0);

    x1 = slice(x0,0,50,1);
    x2 = slice(x0,50,50,1);

    phi = value(0.1);

    z1 = mix(phi,f,x1);
    intermediate(,z1,{First half mixer});
    z2 = mix(phi,f,x2);
```

```
intermediate(,z2,{Second half mixer});

zA = slice(z0,0,50,1);
zB = slice(z0,50,50,1);

rzA = real(zA);
rzB = real(zB);
rz1 = real(z1);
rz2 = real(z2);
izA = imag(zA);
izB = imag(zB);
iz1 = imag(z1);
iz2 = imag(z2);

rdifA = sub(rzA,rz1);
intermediate(,rdifA,{First half real differences});
min(rdifA);
intermediate(,rminA,{Minimum first half real deviation});
max(rdifA);
intermediate(,rmaxA,{Maximum first half real deviation});

rdifB = sub(rzB,rz2);
intermediate(,rdifB,{Second half real differences});
min(rdifB);
intermediate(,rminB,{Minimum second half real deviation});
max(rdifB);
intermediate(,rmaxB,{Maximum second half real deviation});

idifA = sub(izA,iz1);
intermediate(,idifA,{First half imag differences});
min(idifA);
intermediate(,iminA,{Minimum first half imag deviation});
max(idifA);
intermediate(,imaxA,{Maximum first half imag deviation});

idifB = sub(izB,iz2);
intermediate(,idifB,{Second half imag differences});
min(idifB);
intermediate(,iminB,{Minimum second half imag deviation});
max(idifB);
intermediate(,imaxB,{Maximum second half imag deviation});

value(z0);
}
}
```

Pass / Fail

- 6. Verify that the minimum and maximum values of the real and imaginary parts of the first and second half differences are zero to floating point (or double) precision, relative to the rms amplitudes of the input data.

Pass / Fail

- 7. Move the output file to mdc/output/api.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: STATAPI01

Purpose: To test that `std::invalid_argument` exceptions thrown by the `min`, `max`, `mean`, `rms`, `variance`, `skewness`, `kurtosis`, and all statistics methods are caught and recorded at the Tcl layer.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), and integer (I2) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N200XX.sim

where XX is either R4, R8, or I2.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ I2

2. (min) Issue the `ldas` command

```
ldasJob {-name dcapi -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -returnprotocol file:/mdc/output/STATAPI01XX.ilwd
  -returnformat ilwd
  -resultcomment {None}
  -resultname {None}
  -aliases { idata = chan_01:data }
  -algorithms {
    data = slice(idata,0,0,1);
    size(n);
    intermediate(.,ISize,{Input data size});
    min(data);
  }
}
```

Pass / Fail

3. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

4. (max) Issue the LDAS command

```
ldasJob {-name dcapi -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -returnprotocol file:/mdc/output/STATAPI01XX.ilwd
  -returnformat ilwd
  -resultcomment {None}
  -resultname {None}
  -aliases { idata = chan_01:data }
  -algorithms {
    data = slice(idata,0,0,1);
    size(n);
    intermediate(,,ISize,{Input data size});
    max(data);
  }
}
```

Pass / Fail

5. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

6. (mean) Issue the LDAS command

```
ldasJob {-name dcapi -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -returnprotocol file:/mdc/output/STATAPI01XX.ilwd
  -returnformat ilwd
  -resultcomment {None}
  -resultname {None}
  -aliases { idata = chan_01:data }
  -algorithms {
    data = slice(idata,0,0,1);
    size(n);
    intermediate(,,ISize,{Input data size});
    mean(data);
  }
}
```

Pass / Fail

7. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

8. (rms) Issue the LDAS command

```
ldasJob {-name dcapi -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -returnprotocol file:/mdc/output/STATAPI01XX.ilwd
  -returnformat ilwd
  -resultcomment {None}
  -resultname {None}
  -aliases { idata = chan_01:data }
  -algorithms {
    data = slice(idata,0,0,1);
    size(n);
    intermediate(,,ISize,{Input data size});
    rms(data);
  }
}
```

Pass / Fail

9. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

10. (variance) Issue the LDAS command

```
ldasJob {-name dcapi -password **** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -returnprotocol file:/mdc/output/STATAPI01XX.ilwd
  -returnformat ilwd
  -resultcomment {None}
  -resultname {None}
  -aliases { idata = chan_01:data }
  -algorithms {
    data = slice(idata,0,0,1);
    size(n);
    intermediate(,,ISize,{Input data size});
    variance(data);
  }
}
```

Pass / Fail

11. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

12. (skewness) Issue the LDAS command

```
ldasJob {-name dcapi -password ***** -email user@host }
```

```

    { conditionData
      -inputprotocol file:/mdc/input/N200XX.sim
      -returnprotocol file:/mdc/output/STATAPI01XX.ilwd
      -returnformat ilwd
      -resultcomment {None}
      -resultname {None}
      -aliases { idata = chan_01:data }
      -algorithms {
        data = slice(idata,0,0,1);
        size(n);
        intermediate(,ISize,{Input data size});
        skewness(data);
      }
    }
  }

```

Pass / Fail

13. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

14. (kurtosis) Issue the LDAS command

```

ldasJob {-name dcapi -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -returnprotocol file:/mdc/output/STATAPI01XX.ilwd
    -returnformat ilwd
    -resultcomment {None}
    -resultname {None}
    -aliases { idata = chan_01:data }
    -algorithms {
      data = slice(idata,0,0,1);
      size(n);
      intermediate(,ISize,{Input data size});
      kurtosis(data);
    }
  }
}

```

Pass / Fail

15. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

16. (all) Issue the LDAS command

```

ldasJob {-name dcapi -password ***** -email user@host }
  { conditionData

```

```

-inputprotocol file:/mdc/input/N200XX.sim
-returnprotocol file:/mdc/output/STATAPI01XX.ilwd
-returnformat ilwd
-resultcomment {None}
-resultname {None}
-aliases { idata = chan_01:data }
-algorithms {
  data = slice(data,0,0,1);
  size(n);
  intermediate(,ISize,{Input data size});
  all(data);
}
}

```

Pass / Fail

17. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: STATAPI02

Purpose: To test that the statistics actions return correct results for an $N = 100$ array of data.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), and integer (I2) data types.

ENVIRONMENT AND PREREQUISITES

The following data sets are required for this test:

1. Input data:

(a) N200XX.sim

where XX is either R4, R8, or I2. These data sets consist of sequences of random numbers.

2. Expected results:

(a) STATAPI02XX.exp

where XX is R4, R8, or I2. These data sets contain the expected results.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ I2

2. Issue the ldas command

```
ldasJob {-name dcapi -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -returnprotocol file:/mdc/output/STATAPI02XX.ilwd
  -returnformat ilwd
  -resultcomment {all}
  -resultname {all}
  -aliases { idata = chan_01:data }
  -algorithms {
    data = slice(idata,0,100,1);
    size(data);
    intermediate(,size,size);
    min(data);
```

```
        intermediate(, ,min,min);
        max(data);
        intermediate(, ,max,max);
        mean(data);
        intermediate(, ,mean,mean);
        rms(data);
        intermediate(, ,rms,rms);
        variance(data);
        intermediate(, ,variance,variance);
        skewness(data);
        intermediate(, ,skewness,skewness);
        kurtosis(data);
        intermediate(, ,kurtosis,kurtosis);
        all(data);
    }
}
```

Pass / Fail

3. Verify that the size of data is returned correctly by the size action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

4. Verify that min of data is returned correctly by the min action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

5. Verify that the max of data is returned correctly by the max action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

6. Verify that the mean of data is returned correctly the by the mean action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

7. Verify that the rms of data is returned correctly by the rms action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

8. Verify that the variance of data is returned correctly the variance action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

9. Verify that the skewness of data is returned correctly by the skewness action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

10. Verify that the kurtosis of data is returned correctly by the kurtosis action by comparing with the previously calculated result in STATAPI02XX.exp.

Pass / Fail

11. Verify that all the statistics are returned correctly the the all action by comparing with the previously calculated results in STATAPI02XX.exp.

Pass / Fail

12. Move the output file to mdc/output/api.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: LINFILTAPI01

Purpose: To test that `std::invalid_argument` exceptions thrown by the `LinFilt` class (corresponding to zero-length IIR filter coefficients, zero-length FIR filter coefficients, and zero-length input sequence) are caught and recorded at the Tcl layer.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N200XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.
2. (Zero-length IIR filter coefficients) Issue `ldas` command

R4/ R8/ C8/ C16

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTAPI01AXX.ilwd
  -returnformat ilwd
  -resultname {None}
  -resultcomment {None}
  -aliases { xlong = chan_01:data; a = chan_03:data; blong = chan_02:data }
  -algorithms {
    x = slice(xlong,0,100,1);
    b = slice(blong,0,13,1);

    a0 = slice(a,0,0,1);
    y = linfilt(b,a0,x);

    value(y);
  }
}
```

```
}

```

Pass / Fail

3. Verify that a `std::invalid_argument`, indicating zero-length IIR filter coefficients, appears in the appropriate log file.

Pass / Fail

4. (Zero-length FIR filter coefficients) Issue `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTAPI01BXX.ilwd
  -returnformat ilwd
  -resultname {None}
  -resultcomment {None}
  -aliases { xlong = chan_01:data; a = chan_03:data; blong = chan_02:data }
  -algorithms {
    x = slice(xlong,0,100,1);
    b = slice(blong,0,13,1);

    b0 = slice(b,0,0,1);
    y = linfilt(b0,a,x);

    value(y);
  }
}
```

Pass / Fail

5. Verify that a `std::invalid_argument`, indicating zero-length FIR filter coefficients, appears in the appropriate log file.

Pass / Fail

6. (Zero-length input sequence) Issue `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTAPI01CXX.ilwd
  -returnformat ilwd
  -resultname {None}
  -resultcomment {None}
  -aliases { xlong = chan_01:data; a = chan_03:data; blong = chan_02:data }
```

```
-algorithms {  
  x = slice(xlong,0,100,1);  
  b = slice(blong,0,13,1);  
  
  x0 = slice(x,0,0,1);  
  y = linfoilt(b,a,x0);  
  
  value(y);  
}  
}
```

Pass / Fail

- 7. Verify that a `std::invalid_argument`, indicating a zero-length input sequence, appears in the appropriate log file.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: LINFILTAPI02

Purpose: To verify that the `LinFilt` action returns correct results when acting on unbroken and broken input sequences.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following data sets are required for this test:

1. Input data:

(a) N200XX.sim

where XX is either R4, R8, C8, or C16. These data sets consist of sequences of random numbers.

2. Expected results:

(a) LINFILTAPI02AXX.exp

where XX is R4, R8, C8, or C16. These data sets contain the expected results for the `LinFilt` action on unbroken sequences of length $N = 100$.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. (unbroken sequence) Issue the following `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol {
      file:/mdc/input/N200XX.sim
      file:/mdc/input/LINFILTAPI02AXX.exp
    }
    -inputformat ilwd
    -returnprotocol file:/LINFILTAPI02AXX.ilwd
    -returnformat ilwd
    -resultname {y}
    -resultcomment {liner filter output}
```

```

-aliases { raw = chan_01:data; a = chan_03:data; blong = chan_02:data;
          exp = linfiltout:data }
-algorithms {
  x = slice(raw,0,100,1);
  b = slice(blong,0,13,1);
  y = linfilt(b,a,x);

  ry = real(y);
  rexp = real(exp);
  rdif = sub(ry, rexp);
  rmin = min(rdif);
  intermediate(, rmin, {minimum of real difference});
  rmax = max(rdif);
  intermediate(, rmax, {maximum of imag difference});
  rrms = rms(rexp);
  intermediate(, rrms, {root mean square of real expected result});

  iy = imag(y);
  iexp = imag(exp);
  idif = sub(iy, iexp);
  imin = min(idif);
  intermediate(, imin, {minimum of imag difference});
  imax = max(idif);
  intermediate(, imax, {maximum of imag difference});
  irms = rms(iexp);
  intermediate(, irms, {root mean square of imag expected result});

  value(y);
}
}

```

Pass / Fail

3. Verify that the minimum and maximum values of the real and imaginary parts of the differences between the calculated and expected results are zero to floating point (or double) precision, relative to the rms amplitude of the expected results.

Pass / Fail

4. Move the output file to mdc/output/api.

Pass / Fail

5. (broken sequence) Issue the following ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
}

```

```
-returnprotocol file:/LINFILTAPIO2BXX.ilwd
-returnformat ilwd
-resultname {y}
-resultcomment {liner filter output}
-aliases { raw = chan_01:data; a = chan_03:data; blong = chan_02:data }
-algorithms {
    b = slice(blong,0,13,1);

    x = slice(raw,0,100,1);
    y = linfilt(b,a,x);
    intermediate(,y,whole);

    x1 = slice(raw,0,50,1);
    y1 = linfilt(b,a,x1,z);
    intermediate(,y1,{first half});
    x2 = slice(raw,50,50,1);
    y2 = linfilt(b,a,x2,z);
    intermediate(,y2,{second half});

    yA = slice(y,0,50,1);
    yB = slice(y,50,50,1);

    ryA = real(yA);
    ryB = real(yB);
    ry1 = real(y1);
    ry2 = real(y2);
    rd1 = sub(ryA,ry1);
    rd2 = sub(ryB,ry2);
    rmin1 = min(rd1);
    intermediate(,rmin1,{minimum of difference of real first halves});
    rmin2 = min(rd2);
    intermediate(,rmin2,{minimum of difference of real second halves});
    rmax1 = max(rd1);
    intermediate(,rmax1,{maximum of difference of real first halves});
    rmax2 = max(rd2);
    intermediate(,rmax2,{maximum of difference of real second halves});
    rrms1 = rms(ryA);
    intermediate(,rrms1,{root mean square of first half});
    rrms2 = rms(ryB);
    intermediate(,rrms2,{root mean square of second half});

    iyA = imag(yA);
    iyB = imag(yB);
    iy1 = imag(y1);
    iy2 = imag(y2);
    id1 = sub(iyA,iy1);
```

```

    id2 = sub(iyB,iy2);
    imin1 = min(id1);
    intermediate(,imin1,{minimum difference of imag first halves});
    imin2 = min(id2);
    intermediate(,imin2,{minumum difference of imag second halves});
    imax1 = max(id1);
    intermediate(,imax1,{maximum difference of imag first halves});
    imax2 = max(id2);
    intermediate(,imax2,{maximum difference of imag second halves});
    irms1 = rms(iyA);
    intermediate(,irms1,{root mean square of first half});
    irms2 = rms(iyB);
    intermediate(,irms2,{root mean square of second half});

    value(y);
  }
}

```

Pass / Fail

6. Verify that the minimum and maximum values of the real and imaginary parts of the first and second half differences are zero to floating point (or double) precision, relative to the rms amplitudes of the input data.

Pass / Fail

7. Move the output file to mdc/output/api.

Pass / Fail**SUMMARY**

Known faults encountered – list bug IDs: _____

 New faults submitted – list bug IDs: _____

TEST RESULT**Pass / Fail**

Test Case: RESAMPLEAPI01

Purpose: To test that exceptions thrown by the `Resample` class are caught and recorded at the Tcl layer.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N200XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($p < 1$) Issue `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEAPI01AXX.ilwd
    -resultname {none}
    -resultcomment {none}
    -aliases { xlong = chan_01:data }
    -algorithms {
      x = slice(xlong,0,100,1);

      y = resample(0,1,x);

      value(y);
    }
  }
}
```

Pass / Fail

3. Verify that a `std::invalid_argument` error, indicating a value of $p < 1$, appears in the appropriate log file.

Pass / Fail

4. ($q < 1$) Issue ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEAPI01BXX.ilwd
    -resultname {none}
    -resultcomment {none}
    -aliases { xlong = chan_01:data }
    -algorithms {
      x = slice(xlong,0,100,1);

      y = resample(1,0,x);

      value(y);
    }
  }
}
```

Pass / Fail

5. Verify that a `std::invalid_argument` error, indicating a value of $q < 1$, appears in the appropriate log file.

Pass / Fail

6. (Both p and q are > 1) Issue ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEAPI01CXX.ilwd
    -resultname {none}
    -resultcomment {none}
    -aliases { xlong = chan_01:data }
    -algorithms {
      x = slice(xlong,0,100,1);

      y = resample(2,2,x);

      value(y);
    }
  }
}
```

Pass / Fail

7. Verify that an `unimplemented_error` exception, indicating that both p and q are > 1 , appears in the appropriate log file.

Pass / Fail

8. (Input length not equal to a multiple of q) Issue `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEAPI01DXX.ilwd
    -resultname {none}
    -resultcomment {none}
    -aliases { xlong = chan_01:data }
    -algorithms {
      x = slice(xlong,0,13,1);

      y = resample(1,5,x);

      value(y);
    }
  }
}
```

Pass / Fail

9. Verify that an `std::logic_error` exception, indicating an input length not equal to a multiple of q , appears in the appropriate log file.

Pass / Fail

10. (Invalid state length) Issue `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N200XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEAPI01EXX.ilwd
    -resultname {none}
    -resultcomment {none}
    -aliases { xlong = chan_01:data }
    -algorithms {
      x = slice(xlong,0,100,1);
      z = slice(xlong,0,13,1);

      y = resample(1,5,x,z);

      value(y);
    }
  }
}
```

Pass / Fail

- 11. Verify that an `std::invalid_argument` exception, indicating a state of the wrong length, appears in the appropriate log file.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: RESAMPLEAPI02

Purpose: To verify that the `Resample` action returns correct values when acting on unbroken and broken sequences.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following data sets are required for this test:

1. Input data:

(a) N200XX.sim

where XX is either R4, R8, C8, or C16. These data sets consist of sequences of random numbers.

2. Expected results:

(a) RESAMPLEAPI02AXX.exp

(b) RESAMPLEAPI02BXX.exp

where XX is R4, R8, C8, or C16. These data sets contain the expected results on unbroken sequences, for upsample by 5 and downsample by 5, respectively.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. (unbroken sequence, upsample by 5) Issue the following `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol {
      file:/mdc/input/N200XX.sim
      file:/mdc/input/RESAMPLEAPI02AXX.exp
    }
    -inputformat ilwd
    -returnprotocol file:/RESAMPLEAPI02AXX.ilwd
    -returnformat ilwd
    -resultname {upsample}
```

```

-resultcomment {Upsample result}
-aliases { raw = chan_01:data; exp = upsampleout:data }
-algorithms {
  x = slice(raw,0,100,1);

  rx = real(x);
  variance(rx);
  intermediate(,,rvarX,{Variance of real input data});
  ix = imag(x);
  variance(ix);
  intermediate(,,ivarX,{Variance of imag input data});

  y = resample(5,1,x);

  ry = real(y);
  rexp = real(exp);
  rdif = sub(ry,rexp);
  min(rdif);
  intermediate(,,rminD,{Minimum real deviation});
  max(rdif);
  intermediate(,,rmaxD,{Maximum real deviation});
  variance(rdif);
  intermediate(,,rvarD,{Variance of real deviation});
  rms(rexp);
  intermediate(,,rrmsE,{RootMeanSquare of real expected result});

  iy = imag(y);
  iexp = imag(exp);
  idif = sub(iy,iexp);
  min(idif);
  intermediate(,,iminD,{Minimum imag deviation});
  max(idif);
  intermediate(,,imaxD,{Maximum imag deviation});
  variance(idif);
  intermediate(,,ivarD,{Variance of imag deviation});
  rms(iexp);
  intermediate(,,irmsE,{RootMeanSquare of imag expected result});

  value(y);
}
}

```

Pass / Fail

3. Verify that the minimum and maximum values of the real and imaginary parts of the differences between the calculated and expected results are zero to floating point (or double) precision, relative to

the rms amplitude of the expected results.

Pass / Fail

4. Move the output file to mdc/output/api.

Pass / Fail

5. (unbroken sequence, downsample by 5) Issue the following ldas command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol {
    file:/mdc/input/N200XX.sim
    file:/mdc/input/RESAMPLEAPI02BXX.exp
  }
  -inputformat ilwd
  -returnprotocol file:/RESAMPLEAPI02BXX.ilwd
  -returnformat ilwd
  -resultname {downsample}
  -resultcomment {Downsample result}
  -aliases { raw = chan_01:data; exp = upsampleout:data }
  -algorithms {
    x = slice(raw,0,100,1);

    rx = real(x);
    variance(rx);
    intermediate(,,rvarX,{Variance of real input data});
    ix = imag(x);
    variance(ix);
    intermediate(,,ivarX,{Variance of imag input data});

    y = resample(1,5,x);

    ry = real(y);
    rexp = real(exp);
    rdif = sub(ry,rexp);
    min(rdif);
    intermediate(,,rminD,{Minimum real deviation});
    max(rdif);
    intermediate(,,rmaxD,{Maximum real deviation});
    variance(rdif);
    intermediate(,,rvarD,{Variance of real deviation});
    rms(rexp);
    intermediate(,,rrmsE,{RootMeanSquare of real expected result});

    iy = imag(y);
    iexp = imag(exp);
```

```

    idif = sub(iy,iexp);
    min(idif);
    intermediate(,iminD,{Minimum imag deviation});
    max(idif);
    intermediate(,imaxD,{Maximum imag deviation});
    variance(idif);
    intermediate(,ivarD,{Variance of imag deviation});
    rms(iexp);
    intermediate(,irmsE,{RootMeanSquare of imag expected result});

    value(y);
  }
}

```

Pass / Fail

6. Verify that the minimum and maximum values of the real and imaginary parts of the differences between the calculated and expected results are zero to floating point (or double) precision, relative to the rms amplitude of the expected results.

Pass / Fail

7. Move the output file to mdc/output/api.

Pass / Fail

8. (broken sequence, downsample by 5) Issue the following ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
  -returnprotocol file:/RESAMPLEAPI02CXX.ilwd
  -returnformat ilwd
  -resultname {downbroken}
  -resultcomment {Broken downsample result}
  -aliases { raw = chan_01:data }
  -algorithms {
    x0 = slice(raw,0,100,1);
    y0 = resample(1,5,x0);

    x1 = slice(x0,0,50,1);
    x2 = slice(x0,50,50,1);

    y1 = resample(1,5,x1,z);
    intermediate(,y1,{First half downsample x 5});
    y2 = resample(1,5,x2,z);
    intermediate(,y1,{Second half downsample x 5});
  }
}

```

```
yA = slice(y0,0,10,1);
yB = slice(y0,10,10,1);

ryA = real(yA);
ryB = real(yB);
ry1 = real(y1);
ry2 = real(y2);
iyA = imag(yA);
iyB = imag(yB);
iy1 = imag(y1);
iy2 = imag(y2);

rdifA = sub(ryA,ry1);
intermediate(,rdifA,{First half real differences});
min(rdifA);
intermediate(,rminA,{Minimum first half real deviation});
max(rdifA);
intermediate(,rmaxA,{Maximum first half real deviation});

rdifB = sub(ryB,ry2);
intermediate(,rdifB,{Second half real differences});
min(rdifB);
intermediate(,rminB,{Minimum second half real deviation});
max(rdifB);
intermediate(,rmaxB,{Maximum second half real deviation});

idifA = sub(iyA,iy1);
intermediate(,idifA,{First half imag differences});
min(idifA);
intermediate(,iminA,{Minimum first half imag deviation});
max(idifA);
intermediate(,imaxA,{Maximum first half imag deviation});

idifB = sub(iyB,iy2);
intermediate(,idifB,{Second half imag differences});
min(idifB);
intermediate(,iminB,{Minimum second half imag deviation});
max(idifB);
intermediate(,imaxB,{Maximum second half imag deviation});

value(y0);
}
}
```

Pass / Fail

9. Verify that the minimum and maximum values of the real and imaginary parts of the first and second half differences are zero to floating point (or double) precision, relative to the rms amplitudes of the input data.

Pass / Fail

10. Move the output file to mdc/output/api.

Pass / Fail

11. (broken sequence, upsample by 2) Issue the following ldas command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N200XX.sim
  -inputformat ilwd
  -returnprotocol file:/RESAMPLEAPI02DXX.ilwd
  -returnformat ilwd
  -resultname {upbroken}
  -resultcomment {Broken upsample result}
  -aliases { raw = chan_01:data }
  -algorithms {
    x0 = slice(raw,0,20,1);
    y0 = resample(2,1,x0);

    x1 = slice(x0,0,10,1);
    x2 = slice(x0,10,10,1);

    y1 = resample(2,1,x1,z);
    intermediate(,y1,{First half upsample x 2});
    y2 = resample(2,1,x2,z);
    intermediate(,y1,{Second half upsample x 2});

    yA = slice(y0,0,20,1);
    yB = slice(y0,20,20,1);

    ryA = real(yA);
    ryB = real(yB);
    ry1 = real(y1);
    ry2 = real(y2);
    iyA = imag(yA);
    iyB = imag(yB);
    iy1 = imag(y1);
    iy2 = imag(y2);

    rdifA = sub(ryA,ry1);
    intermediate(,rdifA,{First half real differences});
    min(rdifA);
```

```

    intermediate(,rminA,{Minimum first half real deviation});
    max(rdifA);
    intermediate(,rmaxA,{Maximum first half real deviation});

    rdifB = sub(ryB,ry2);
    intermediate(,rdifB,{Second half real differences});
    min(rdifB);
    intermediate(,rminB,{Minimum second half real deviation});
    max(rdifB);
    intermediate(,rmaxB,{Maximum second half real deviation});

    idifA = sub(iyA,iy1);
    intermediate(,idifA,{First half imag differences});
    min(idifA);
    intermediate(,iminA,{Minimum first half imag deviation});
    max(idifA);
    intermediate(,imaxA,{Maximum first half imag deviation});

    idifB = sub(iyB,iy2);
    intermediate(,idifB,{Second half imag differences});
    min(idifB);
    intermediate(,iminB,{Minimum second half imag deviation});
    max(idifB);
    intermediate(,imaxB,{Maximum second half imag deviation});

    value(y0);
  }
}

```

Pass / Fail

12. Verify that the minimum and maximum values of the real and imaginary parts of the first and second half differences are zero to floating point (or double) precision, relative to the rms amplitudes of the input data.

Pass / Fail

13. Move the output file to mdc/output/api.

Pass / Fail**SUMMARY**

Known faults encountered – list bug IDs: _____

 New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: FFT01

Purpose: To test that `std::invalid_argument` exceptions thrown by the FFT method when acting on sequences with invalid lengths ($N = 0$ and $N = \text{MaximumFFTLength} + 1 = 8388609$) are caught and recorded at the Tcl layer.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388609XXFFT.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.
2. ($N = 0$) Issue the LDAS command

R4/ R8/ C8/ C16

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388609XXFFT.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/FFTAPI01AXX.ilwd
    -returnformat ilwd
    -aliases { idata = chan_01:data }
    -algorithms {
      x = slice(idata,0,0,1);
      rx = real(x);
      n = size(rx);
      intermediate(.,n,{length of input data});
      y=fft(x);
    }
    -resultcomment {none}
    -resultname {none}
  }
}
```

Pass / Fail

3. Verify that a `std::invalid_argument` error appears in the appropriate log file.

Pass / Fail

4. ($N = 8388609$) Issue the LDAS command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388609XXFFT.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/FFTAPIO1BXX.ilwd
    -returnformat ilwd
    -aliases { idata = chan_01:data }
    -algorithms {
      x = slice(idata,0,8388609,1);
      rx = real(x);
      n = size(rx);
      intermediate(.,n,{length of input data});
      y=fft(x);
    }
    -resultcomment {none}
    -resultname {none}
  }
}
```

Pass / Fail

5. Verify that a `std::invalid_argument` error appears in the appropriate log file.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: FFT02

Purpose: To verify that the FFT and IFFT functions return correct results for a range of valid input lengths: $N = 1$, 1024, 122880, and 1843200; and to verify that Parseval's theorem holds for an FFT of length $N = \text{MaximumFFTLength} = 8388608$.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following data sets are required for this test:

1. Input data for direct comparison tests:

(a) N8388608XX.sim

where XX is either R4, R8, C8, or C16. These data sets consist of sequences of random numbers.

2. Expected results for direct comparison tests:

(a) FFTAPI02AXX.exp ($N = 1$)

(b) FFTAPI02BXX.exp ($N = 1024$)

(c) FFTAPI02CXX.exp ($N = 122880$)

(d) FFTAPI02DXX.exp ($N = 1843200$)

where XX is R4, R8, C8, or C16. These data sets contain the expected results for FFTs of variable length data.

3. Input data for Parseval's theorem test: ($N = 8388608$)

(a) N8388608XXFFT.sim

These data sets consist of sequences of all 1's (real) or all $1 + i$'s (complex).

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

PROCEDURE

1. ($N = 1$) Issue the LDAS command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol {
    file:/mdc/input/N8388608XX.sim
    file:/mdc/input/FFTAPIO2AXX.exp
  }
  -inputformat ilwd
  -returnprotocol file:/FFTAPIO2AXX.ilwd
  -returnformat ilwd
  -aliases { idata = chan_01:data; exp = fftout:data }
  -algorithms {
    x = slice(idata,0,1,1);
    rx = real(x);
    n1 = size(rx);
    intermediate(,n1,{length of input data before FFT});
    y = fft(x);

    ry = real(y);
    rexp = real(exp);
    rdif1 = sub(ry,rexp);
    rmin1 = min(rdif1);
    intermediate(,rmin1,{minimum of real difference for FFT});
    rmax1 = max(rdif1);
    intermediate(,rmax1,{maximum of real difference for FFT});

    iy = imag(y);
    iexp = imag(exp);
    idif1 = sub(iy,iexp);
    imin1 = min(idif1);
    intermediate(,imin1,{minimum of imag difference for FFT});
    imax1 = max(idif1);
    intermediate(,imax1,{maximum of imag difference for FFT});

    rrms1 = rms(ry);
    intermediate(,rrms1,{rms of real FFT output});
    irms1 = rms(iy);
    intermediate(,irms1,{rms of imag FFT output});

    z = ifft(y);
    rz = real(z);
    n2 = size(rz);
    intermediate(,n2,{length of output data after IFFT});

    rdif2 = sub(rz,rx);

```

```

    rmin2 = min(rdif2);
    intermediate(,rmin2,{minimum of real difference for IFFT});
    rmax2 = max(rdif2);
    intermediate(,rmax2,{maximum of real difference for IFFT});

    iz = imag(z);
    ix = imag(x);
    idif2 = sub(iz,ix);
    imin2 = min(idif2);
    intermediate(,imin2,{minimum of imag difference for IFFT});
    imax2 = max(idif2);
    intermediate(,imax2,{maximum of imag difference for IFFT});

    rrms2 = rms(rz);
    intermediate(,rrms2,{rms of real IFFT output});
    irms2 = rms(iz);

}
-resultname {irms2}
-resultcomment {rms of imag IFFT output}
}

```

Pass / Fail

2. Verify that the lengths of the data before the FFT and after the IFFT are equal to 1.

Pass / Fail

3. Verify that the minimum and maximum values of the real and imaginary parts of the differences for the FFT and IFFT are zero to floating point (or double) precision relative to the rms amplitude of the FFT and IFFT values.

Pass / Fail

4. Move the output file to mdc/output/api.

Pass / Fail

5. ($N = 1024$) Issue the LDAS command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol {
    file:/mdc/input/N8388608XX.sim
    file:/mdc/input/FFTAPI02BXX.exp
  }
  -inputformat ilwd
  -returnprotocol file:/FFTAPI02BXX.ilwd
  -returnformat ilwd
  -aliases { idata = chan_01:data; exp = fftout:data }
}

```

```
-algorithms {
  x = slice(idata,0,1024,1);
  rx = real(x);
  n1 = size(rx);
  intermediate(,n1,{length of input data before FFT});
  y = fft(x);

  ry = real(y);
  rexp = real(exp);
  rdif1 = sub(ry,rexp);
  rmin1 = min(rdif1);
  intermediate(,rmin1,{minimum of real difference for FFT});
  rmax1 = max(rdif1);
  intermediate(,rmax1,{maximum of real difference for FFT});

  iy = imag(y);
  iexp = imag(exp);
  idif1 = sub(iy,iexp);
  imin1 = min(idif1);
  intermediate(,imin1,{minimum of imag difference for FFT});
  imax1 = max(idif1);
  intermediate(,imax1,{maximum of imag difference for FFT});

  rrms1 = rms(ry);
  intermediate(,rrms1,{rms of real FFT output});
  irms1 = rms(iy);
  intermediate(,irms1,{rms of imag FFT output});

  z = ifft(y);
  rz = real(z);
  n2 = size(rz);
  intermediate(,n2,{length of output data after IFFT});

  rdif2 = sub(rz,rx);
  rmin2 = min(rdif2);
  intermediate(,rmin2,{minimum of real difference for IFFT});
  rmax2 = max(rdif2);
  intermediate(,rmax2,{maximum of real difference for IFFT});

  iz = imag(z);
  ix = imag(x);
  idif2 = sub(iz,ix);
  imin2 = min(idif2);
  intermediate(,imin2,{minimum of imag difference for IFFT});
  imax2 = max(idif2);
  intermediate(,imax2,{maximum of imag difference for IFFT});
```

```

rrms2 = rms(rz);
intermediate(,rrms2,{rms of real IFFT output});
irms2 = rms(iz);

}
-resultname {irms2}
-resultcomment {rms of imag IFFT output}
}

```

Pass / Fail

6. Verify that the lengths of the data before the FFT and after the IFFT are equal to 1024.

Pass / Fail

7. Verify that the minimum and maximum values of the real and imaginary parts of the differences for the FFT and IFFT are zero to floating point (or double) precision relative to the rms amplitude of the FFT and IFFT values.

Pass / Fail

8. Move the output file to mdc/output/api.

Pass / Fail

9. ($N = 122880$) Issue the LDAS command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol {
    file:/mdc/input/N8388608XX.sim
    file:/mdc/input/FFTAPI02CXX.exp
  }
  -inputformat ilwd
  -returnprotocol file:/FFTAPI02CXX.ilwd
  -returnformat ilwd
  -aliases { idata = chan_01:data; exp = fftout:data }
  -algorithms {
    x = slice(idata,0,122880,1);
    rx = real(x);
    n1 = size(rx);
    intermediate(,n1,{length of input data before FFT});
    y = fft(x);

    ry = real(y);
    rexp = real(exp);
    rdif1 = sub(ry,rexp);
    rmin1 = min(rdif1);
    intermediate(,rmin1,{minimum of real difference for FFT});
  }
}

```

```
    rmax1 = max(rdif1);
    intermediate(,rmax1,{maximum of real difference for FFT});

    iy = imag(y);
    iexp = imag(exp);
    idif1 = sub(iy,iexp);
    imin1 = min(idif1);
    intermediate(,imin1,{minimum of imag difference for FFT});
    imax1 = max(idif1);
    intermediate(,imax1,{maximum of imag difference for FFT});

    rrms1 = rms(ry);
    intermediate(,rrms1,{rms of real FFT output});
    irms1 = rms(iy);
    intermediate(,irms1,{rms of imag FFT output});

    z = ifft(y);
    rz = real(z);
    n2 = size(rz);
    intermediate(,n2,{length of output (data after IFFT)});

    rdif2 = sub(rz,rx);
    rmin2 = min(rdif2);
    intermediate(,rmin2,{minimum of real difference for IFFT});
    rmax2 = max(rdif2);
    intermediate(,rmax2,{maximum of real difference for IFFT});

    iz = imag(z);
    ix = imag(x);
    idif2 = sub(iz,ix);
    imin2 = min(idif2);
    intermediate(,imin2,{minimum of imag difference for IFFT});
    imax2 = max(idif2);
    intermediate(,imax2,{maximum of imag difference for IFFT});

    rrms2 = rms(rz);
    intermediate(,rrms2,{rms of real IFFT output});
    irms2 = rms(iz);

}
-resultname {irms2}
-resultcomment {rms of imag IFFT output}
}
```

Pass / Fail

10. Verify that the lengths of the data before the FFT and after the IFFT are equal to 122880. **Pass / Fail**
11. Verify that the minimum and maximum values of the real and imaginary parts of the differences for the FFT and IFFT are zero to floating point (or double) precision relative to the rms amplitude of the FFT and IFFT values. **Pass / Fail**
12. Move the output file to mdc/output/api. **Pass / Fail**
13. ($N = 1843200$) Issue the LDAS command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol {
    file:/mdc/input/N8388608XX.sim
    file:/mdc/input/FFTAPIO2DXX.exp
  }
  -inputformat ilwd
  -returnprotocol file:/FFTAPIO2DXX.ilwd
  -returnformat ilwd
  -aliases { idata = chan_01:data; exp = fftout:data }
  -algorithms {
    x = slice(idata,0,1843200,1);
    rx = real(x);
    n1 = size(rx);
    intermediate(,n1,{length of input data before FFT});
    y = fft(x);

    ry = real(y);
    rexp = real(exp);
    rdif1 = sub(ry, rexp);
    rmin1 = min(rdif1);
    intermediate(,rmin1,{minimum of real difference for FFT});
    rmax1 = max(rdif1);
    intermediate(,rmax1,{maximum of real difference for FFT});

    iy = imag(y);
    iexp = imag(exp);
    idif1 = sub(iy, iexp);
    imin1 = min(idif1);
    intermediate(,imin1,{minimum of imag difference for FFT});
    imax1 = max(idif1);
    intermediate(,imax1,{maximum of imag difference for FFT});

    rrms1 = rms(ry);
```

```

intermediate(,rrms1,{rms of real FFT output});
irms1 = rms(iy);
intermediate(,irms1,{rms of imag FFT output});

z = ifft(y);
rz = real(z);
n2 = size(rz);
intermediate(,n2,{length of output data after IFFT});

rdif2 = sub(rz,rx);
rmin2 = min(rdif2);
intermediate(,rmin2,{minimum of real difference for IFFT});
rmax2 = max(rdif2);
intermediate(,rmax2,{maximum of real difference for IFFT});

iz = imag(z);
ix = imag(x);
idif2 = sub(iz,ix);
imin2 = min(idif2);
intermediate(,imin2,{minimum of imag difference for IFFT});
imax2 = max(idif2);
intermediate(,imax2,{maximum of imag difference for IFFT});

rrms2 = rms(rz);
intermediate(,rrms2,{rms of real IFFT output});
irms2 = rms(iz);

}
-resultname {irms2}
-resultcomment {rms of imag IFFT output}
}

```

Pass / Fail

14. Verify that the lengths of the data before the FFT and after the IFFT are equal to 1843200.

Pass / Fail

15. Verify that the minimum and maximum values of the real and imaginary parts of the differences for the FFT and IFFT are zero to floating point (or double) precision relative to the rms amplitude of the FFT and IFFT values.

Pass / Fail

16. Move the output file to mdc/output/api.

Pass / Fail

17. (Parseval's theorem test for $N = 8388608$) Issue the LDAS command

```

ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XXFFT.sim
    -inputformat ilwd
    -returnprotocol file:/FFTAPI02EXX.ilwd
    -returnformat ilwd
    -aliases { idata = chan_01:data }
    -algorithms {
      x = slice(idata,0,8388608,1);
      rx = real(x);
      n = size(rx);
      intermediate(,n,{length of input data before FFT});
      y = fft(x);

      ry = real(y);
      rdc = slice(ry,4194304,1,1);
      value(rdc);
      intermediate(,rdc,{DC value of real FFT output});
      rmin = min(ry);
      intermediate(,rmin,{minimum of real FFT output});
      rmax = max(ry);
      intermediate(,rmax,{maximum of real FFT output});
      rrms = rms(ry);
      intermediate(,rrms,{rms of real FFT output});

      iy = imag(y);
      idc = slice(iy,4194304,1,1);
      value(idc);
      intermediate(,idc,{DC value of imag FFT output});
      imin = min(iy);
      intermediate(,imin,{minimum of imag FFT output});
      imax = max(iy);
      intermediate(,imax,{maximum of imag FFT output});
      irms = rms(iy);
    }
    -resultname {irms}
    -resultcomment {rms of imag FFT output}
  }
}

```

Pass / Fail

18. Verify that the length of the data before the FFT is equal to 8388608.

Pass / Fail

19. Verify that the DC and maximum values of the real part of the FFT output are equal to 8388608 to floating point (or double) precision.

Pass / Fail

20. Verify that the rms amplitude of the real part of the FFT output is equal to $\sqrt{8388608} = 2896.309\dots$ to floating point (or double) precision.

Pass / Fail

21. Verify that minimum value of the real part of the FFT output is zero to floating point (or double) precision relative to the rms amplitude of the real part of the FFT output.

Pass / Fail

22. Verify that the DC, minimum, maximum, and rms values of the imaginary part of the FFT output are zero to floating point (or double) precision relative to the rms amplitude of the imaginary part of the FFT output.

Pass / Fail

23. Move the output file to `mdc/output/api`.

Pass / Fail

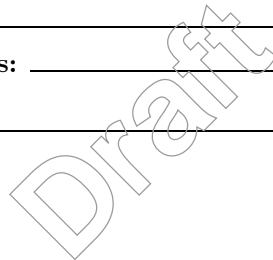
SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail



Test Case: PSDEstimate01

Purpose: To test that `std::invalid_argument` exceptions thrown by the Welch estimate object for a zero length sequence, `fft length = 0`, `overlap length = fft length`, and `overlap length > fft length` are caught and recorded at the Tcl layer.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N10XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.
2. (Zero length sequence) Issue the `ldas` command

R4/ R8/ C8/ C16

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N10XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/PSDAPIO1AXX.ilwd
  -returnformat ilwd
  -resultname {none}
  -resultcomment {none}
  -aliases {data=chan_01:data }
  -algorithms {
    x = slice(data, 0, 0, 1);
    intermediate(,chan_01:data,{slice of input data});
    rx = real(x);
    n = size(rx);
    intermediate(,size,{size of input data});
    spec = psd(freq, x);
  }
}
```

```
}

```

Pass / Fail

3. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

4. (fft length = 0) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N10XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/PSDAPIO1BXX.ilwd
    -returnformat ilwd
    -resultname {none}
    -resultcomment {none}
    -aliases {data=chan_01:data }
    -algorithms {
      x = slice(data, 0, 1024, 1);
      intermediate(,chan_01:data,{slice of input data});
      rx = real(x);
      n = size(rx);
      intermediate(,size,{size of input data});
      spec = psd(freq, x, 0);
    }
  }
}
```

Pass / Fail

5. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

6. (overlap length = fft length) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N10XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/PSDAPIO1CXX.ilwd
    -returnformat ilwd
    -resultname {none}
    -resultcomment {none}
    -aliases {data=chan_01:data }
    -algorithms {
      x = slice(data, 0, 1024, 1);

```

```

        intermediate(,chan_01:data,{slice of input data});
        rx = real(x);
        n = size(rx);
        intermediate(,size,{size of input data});
        spec = psd(freq, x, 1024, _, 1024);
    }
}

```

Pass / Fail

7. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

8. (overlap length > fft length) Issue the `ldas` command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N10XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/PSDAPI01DXX.ilwd
  -returnformat ilwd
  -resultname {none}
  -resultcomment {none}
  -aliases {data=chan_01:data }
  -algorithms {
    x = slice(data, 0, 1024, 1);
    intermediate(,chan_01:data,{slice of input data});
    rx = real(x);
    n = size(rx);
    intermediate(,size,{size of input data});
    spec = psd(freq, x, 1024, _, 1025);
  }
}

```

Pass / Fail

9. Verify that a `std::invalid_argument` exception appears in the appropriate log file.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: PSDEstimate02

Purpose: To verify that the Welch estimate apply method returns correct results for the power spectrum estimate and discrete frequency values for different choices of the fft length and overlap length.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following data sets are required for this test:

1. Input data:

- (a) N10XX.sim

where XX is either R4, R8, C8, or C16. These data sets consist of sequences of random numbers.

2. Expected results:

- (a) PSDAPI02AXX.exp (default fft length, default overlap length)
 (b) PSDAPI02BXX.exp (fft length = 256, overlap length = 0)
 (c) PSDAPI02CXX.exp (fft length = 256, overlap length = 128)

where XX is R4, R8, C8, or C16. These data sets contain the expected results for different power spectrum estimates (defined by different fft lengths and overlap lengths) for input data of length $N = 1024$.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. (default fft length, default overlap length) Issue the following ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol {
      file:/mdc/input/N10XX.sim
      file:/mdc/input/PSDAPI02AXX.exp
    }
  }
```

```
-inputformat ilwd
-returnprotocol file:/PSDAPI02AXX.ilwd
-returnformat ilwd
-aliases {data=chan_01:data; freqexp=freqout:data; specexp=specout:data}
-algorithms {
  x = slice(data, 0, 1024, 1);
  intermediate(,chan_01:data,{slice of input data});

  spec = psd(freq, x);
  intermediate(,spec,{spec});

  value(specexp);
  intermediate(,specexp,{specexp});

  specdif = sub(spec, specexp);
  intermediate(,specdif,{diff of spec and specexp});

  specmin = min(specdif);
  intermediate(,specmin,{minimum of difference});

  specmax = max(specdif);
  intermediate(,specmax,{maximum of difference});

  specrms = rms(specexp);
  intermediate(,specrms,{root mean square of expected result});

  value(freq);
  intermediate(,freq,{freq});

  value(freqexp);
  intermediate(,freqexp,{freqexp});

  freqdif = sub(freq, freqexp);
  intermediate(,freqdif,{freqdif});

  freqmin = min(freqdif);
  intermediate(,freqmin,{minimum of difference});

  freqmax = max(freqdif);
  intermediate(,freqmax,{maximum of difference});

  freqrms = rms(freqexp);
}
-resultname {freqrms}
-resultcomment {root mean square of expected result}
}
```

Pass / Fail

3. Verify that the minimum and maximum values of the differences for the power spectrum estimate and discrete frequency values are zero to floating point (or double) precision relative to the rms amplitude of the expected power spectrum estimate and discrete frequency values.

Pass / Fail

4. Move the output file to mdc/output/api.

Pass / Fail

5. (fft length = 256, overlap length = 0) Issue the following ldas command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol {
    file:/mdc/input/N10XX.sim
    file:/mdc/input/PSDAPI02BXX.exp
  }
  -inputformat ilwd
  -returnprotocol file:/PSDAPI02BXX.ilwd
  -returnformat ilwd
  -aliases {data=chan_01:data; freqexp=freqout:data; specexp=specout:data}
  -algorithms {
    x = slice(data, 0, 1024, 1);
    intermediate(,chan_01:data,{slice of input data});

    spec = psd(freq, x, 256, _, 0);
    intermediate(,spec,{spec});

    value(specexp);
    intermediate(,specexp,{specexp});

    specdif = sub(spec, specexp);
    intermediate(,specdif,{diff of spec and specexp});

    specmin = min(specdif);
    intermediate(,specmin,{minimum of difference});

    specmax = max(specdif);
    intermediate(,specmax,{maximum of difference});

    specrms = rms(specexp);
    intermediate(,specrms,{root mean square of expected result});

    value(freq);
    intermediate(,freq,{freq});
```

```

    value(freqexp);
    intermediate(,freqexp,{freqexp});

    freqdif = sub(freq, freqexp);
    intermediate(,freqdif,{freqdif});

    freqmin = min(freqdif);
    intermediate(,freqmin,{minimum of difference});

    freqmax = max(freqdif);
    intermediate(,freqmax,{maximum of difference});

    freqrms = rms(freqexp);
}
-resultname {freqrms}
-resultcomment {root mean square of expected result}
}

```

Pass / Fail

6. Verify that the minimum and maximum values of the differences for the power spectrum estimate and discrete frequency values are zero to floating point (or double) precision relative to the rms amplitude of the expected power spectrum estimate and discrete frequency values.

Pass / Fail

7. Move the output file to mdc/output/api.

Pass / Fail

8. (fft length = 256, overlap length = 128) Issue the following ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol {
    file:/mdc/input/N10XX.sim
    file:/mdc/input/PSDAPIO2CXX.exp
  }
  -inputformat ilwd
  -returnprotocol file:/PSDAPIO2CXX.ilwd
  -returnformat ilwd
  -aliases {data=chan_01:data; freqexp=freqout:data; specexp=specout:data}
  -algorithms {
    x = slice(data, 0, 1024, 1);
    intermediate(,chan_01:data,{slice of input data});

    spec = psd(freq, x, 256, _, 128, _);
    intermediate(,spec,{spec});
  }
}

```

```

value(specexp);
intermediate(,specexp,{specexp});

specdif = sub(spec, specexp);
intermediate(,specdif,{diff of spec and specexp});

specmin = min(specdif);
intermediate(,specmin,{minimum of difference});

specmax = max(specdif);
intermediate(,specmax,{maximum of difference});

specrms = rms(specexp);
intermediate(,specrms,{root mean square of expected result});

value(freq);
intermediate(,freq,{freq});

value(freqexp);
intermediate(,freqexp,{freqexp});

freqdif = sub(freq, freqexp);
intermediate(,freqdif,{freqdif});

freqmin = min(freqdif);
intermediate(,freqmin,{minimum of difference});

freqmax = max(freqdif);
intermediate(,freqmax,{maximum of difference});

freqrms = rms(freqexp);
}
-resultname {freqrms}
-resultcomment {root mean square of expected result}
}

```

Pass / Fail

9. Verify that the minimum and maximum values of the differences for the power spectrum estimate and discrete frequency values are zero to floating point (or double) precision relative to the rms amplitude of the expected power spectrum estimate and discrete frequency values.

Pass / Fail

10. Move the output file to mdc/output/api.

Pass / Fail**SUMMARY**

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft


```

    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

3. Record the mean time required to mix a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 2048 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,2048,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

5. Record the mean time required to mix a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/MIXERPERF01XX-4096.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for mixing size 4096 sequence}
    -aliases { xraw = chan_01:data }
    -algorithms {
      f = value(0.12345);
      phi = value(0.0);
      x = slice(xraw,0,4096,1);
      t00 = user_time();
      intermediate(,t00,start utime);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      t32 = user_time();
      intermediate(,t32,end utime);
      t32 = sub(t32,t00);
      intermediate(,t32,raw difference);
      time = div(t32,32);
    }
  }
}
```

Pass / Fail

7. Record the mean time required to mix a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
```

```

-inputprotocol file:/mdc/input/N8388608XX.sim
-inputformat ilwd
-returnprotocol file:/mdc/output/MIXERPERF01XX-8192.ilwd
-returnformat ilwd
-resultname {time}
-resultcomment {time for mixing size 8192 sequence}
-aliases { xraw = chan_01:data }
-algorithms {
  f = value(0.12345);
  phi = value(0.0);
  x = slice(xraw,0,8192,1);
  t00 = user_time();
  intermediate(,t00,start utime);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}

```

Pass / Fail

9. Record the mean time required to mix a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 16384 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);

```

```

    phi = value(0.0);
    x = slice(xraw,0,16384,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

11. Record the mean time required to mix a data sequence of length 2^{14} .

Pass / Fail

12. ($N = 2^{15}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-32768.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 32768 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,32768,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  }
}

```

```

    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

13. Record the mean time required to mix a data sequence of length 2^{15} .

Pass / Fail

14. ($N = 2^{16}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-65536.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 65536 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,65536,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

```

}
```

Pass / Fail

15. Record the mean time required to mix a data sequence of length 2^{16} .

Pass / Fail

16. ($N = 2^{17}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/MIXERPERF01XX-131072.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for mixing size 131072 sequence}
    -aliases { xraw = chan_01:data }
    -algorithms {
      f = value(0.12345);
      phi = value(0.0);
      x = slice(xraw,0,131072,1);
      t00 = user_time();
      intermediate(,t00,start utime);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
      t32 = user_time();
      intermediate(,t32,end utime);
      t32 = sub(t32,t00);
      intermediate(,t32,raw difference);
      time = div(t32,32);
    }
  }
}
```

Pass / Fail

17. Record the mean time required to mix a data sequence of length 2^{17} .

Pass / Fail

18. ($N = 2^{18}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-262144.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 262144 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,262144,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

19. Record the mean time required to mix a data sequence of length 2^{18} .

Pass / Fail

20. ($N = 2^{19}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-524288.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 524288 sequence}
  -aliases { xraw = chan_01:data }

```

```

-algorithms {
  f = value(0.12345);
  phi = value(0.0);
  x = slice(xraw,0,524288,1);
  t00 = user_time();
  intermediate(,t00,start utime);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}

```

Pass / Fail

21. Record the mean time required to mix a data sequence of length 2^{19} .

Pass / Fail

22. ($N = 2^{20}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-1048576.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 1048576 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,1048576,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
  }
}

```

```

    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

23. Record the mean time required to mix a data sequence of length 2^{20} .

Pass / Fail

24. ($N = 2^{21}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-2097152.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 2097152 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,2097152,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
  }
}

```

```

        time = div(t32,32);
    }
}

```

Pass / Fail

25. Record the mean time required to mix a data sequence of length 2^{21} .

Pass / Fail

26. ($N = 2^{22}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-4194304.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 4194304 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,4194304,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

27. Record the mean time required to mix a data sequence of length 2^{22} .

Pass / Fail

28. ($N = 2^{23}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/MIXERPERF01XX-8388608.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for mixing size 8388608 sequence}
  -aliases { xraw = chan_01:data }
  -algorithms {
    f = value(0.12345);
    phi = value(0.0);
    x = slice(xraw,0,8388608,1);
    t00 = user_time();
    intermediate(,t00,start utime);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x); y = mix(phi,f,x);
    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}

```

Pass / Fail

29. Record the mean time required to mix a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to mix a data sequence vs. its length.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: LINFILTPERF01

Purpose: Generate performance statistics for the `linfilt` action using an FIR filter.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($N = 2^{10}$) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF01XX-1024.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {FIR linear filter on size 1024 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1024,1);
    a = slice(araw,0,1,1);
    b = slice(brow,0,13,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
  }
}
```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

```

```

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);

```

```

}
}

```

Pass / Fail

3. Record the mean time required to (FIR) linear filter a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF01XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {FIR linear filter on size 2048 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,2048,1);
    a = slice(araw,0,1,1);
    b = slice(brow,0,13,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);

```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(, t32, end utime);
t32 = sub(t32, t00);
intermediate(, t32, raw difference);
time = div(t32, 32);
}
}

```

Pass / Fail

5. Record the mean time required to (FIR) linear filter a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF01XX-4096.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {FIR linear filter on size 4096 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,4096,1);
    a = slice(araw,0,1,1);
    b = slice(braw,0,13,1);

    t00 = user_time();

```

```

intermediate(,t00,start utime);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);
}
}

```

Pass / Fail

7. Record the mean time required to (FIR) linear filter a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF01XX-8192.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {FIR linear filter on size 8192 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,8192,1);
    a = slice(araw,0,1,1);
  }
}

```

```

    b = slice(braw,0,13,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);

}
}

```

Pass / Fail

9. Record the mean time required to (FIR) linear filter a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF01XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {FIR linear filter on size 16384 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; braw = chan_02:data }
}

```



```

-resultname {time}
-resultcomment {FIR linear filter on size 32768 sequence}
-aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
-algorithms {
  x = slice(xraw,0,32768,1);
  a = slice(araw,0,1,1);
  b = slice(brow,0,13,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}

```

Pass / Fail

13. Record the mean time required to (FIR) linear filter a data sequence of length 2^{15} .

Pass / Fail

14. ($N = 2^{16}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
}

```



```

    }
}

```

Pass / Fail

21. Record the mean time required to (FIR) linear filter a data sequence of length 2^{19} .

Pass / Fail

22. ($N = 2^{20}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF01XX-1048576.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {FIR linear filter on size 1048576 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);
    a = slice(araw,0,1,1);
    b = slice(brow,0,13,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);

```



```

    y = linsfilt(b,a,x); y = linsfilt(b,a,x);
    y = linsfilt(b,a,x); y = linsfilt(b,a,x);
    y = linsfilt(b,a,x); y = linsfilt(b,a,x);
    y = linsfilt(b,a,x); y = linsfilt(b,a,x);
    y = linsfilt(b,a,x); y = linsfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
}
}

```

Pass / Fail

29. Record the mean time required to (FIR) linear filter a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to (FIR) linear filter a data sequence vs. its length.

Pass / Fail**SUMMARY**

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT**Pass / Fail**

Test Case: LINFILTPERF02

Purpose: Generate performance statistics for the `linfilt` action using an IIR filter.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($N = 2^{10}$) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF02XX-1024.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {IIR linear filter on size 1024 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1024,1);
    a = slice(araw,0,13,1);
    b = slice(brow,0,1,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
  }
```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);

}
}

```

Pass / Fail

3. Record the mean time required to (IIR) linear filter a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF02XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {IIR linear filter on size 2048 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,2048,1);
    a = slice(araw,0,13,1);
    b = slice(braw,0,1,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);

```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);
}
}

```

Pass / Fail

5. Record the mean time required to (IIR) linear filter a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF02XX-4096.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {IIR linear filter on size 4096 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,4096,1);
    a = slice(araw,0,13,1);
    b = slice(braw,0,1,1);

    t00 = user_time();

```

```

intermediate(,t00,start utime);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);
}
}

```

Pass / Fail

7. Record the mean time required to (IIR) linear filter a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF02XX-8192.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {IIR linear filter on size 8192 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,8192,1);
    a = slice(araw,0,13,1);
  }
}

```

```

    b = slice(braw,0,1,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);

}
}

```

Pass / Fail

9. Record the mean time required to (IIR) linear filter a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF02XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {IIR linear filter on size 16384 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; braw = chan_02:data }
}

```



```

-resultname {time}
-resultcomment {IIR linear filter on size 32768 sequence}
-aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
-algorithms {
  x = slice(xraw,0,32768,1);
  a = slice(araw,0,13,1);
  b = slice(braw,0,1,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}

```

Pass / Fail

13. Record the mean time required to (IIR) linear filter a data sequence of length 2^{15} .

Pass / Fail

14. ($N = 2^{16}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
}

```

```

-inputformat ilwd
-returnprotocol file:/mdc/output/LINFILTPERF02XX-65536.ilwd
-returnformat ilwd
-resultname {time}
-resultcomment {IIR linear filter on size 65536 sequence}
-aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
-algorithms {
  x = slice(xraw,0,65536,1);
  a = slice(araw,0,13,1);
  b = slice(braw,0,1,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}

```

Pass / Fail

15. Record the mean time required to (IIR) linear filter a data sequence of length 2^{16} .

Pass / Fail

16. ($N = 2^{17}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF02XX-131072.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {IIR linear filter on size 131072 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,131072,1);
    a = slice(araw,0,13,1);
    b = slice(braw,0,1,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);
  }
}
```

Pass / Fail

17. Record the mean time required to (IIR) linear filter a data sequence of length 2^{17} .


```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);

}
}

```

Pass / Fail

29. Record the mean time required to (IIR) linear filter a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to (IIR) linear filter a data sequence vs. its length.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: LINFILTPERF03

Purpose: Generate performance statistics for the `linfilt` action using a pole-zero filter.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($N = 2^{10}$) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF03XX-1024.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {pole-zero linear filter on size 1024 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1024,1);
    a = slice(araw,0,13,1);
    b = slice(braw,0,13,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
  }
```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);
}
}

```

Pass / Fail

3. Record the mean time required to (pole-zero) linear filter a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF03XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {pole-zero linear filter on size 2048 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,2048,1);
    a = slice(araw,0,13,1);
    b = slice(braw,0,13,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);

```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);
}
}

```

Pass / Fail

5. Record the mean time required to (pole-zero) linear filter a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
-inputprotocol file:/mdc/input/N8388608XX.sim
-inputformat ilwd
-returnprotocol file:/mdc/output/LINFILTPERF03XX-4096.ilwd
-returnformat ilwd
-resultname {time}
-resultcomment {pole-zero linear filter on size 4096 sequence}
-aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
-algorithms {
x = slice(xraw,0,4096,1);
a = slice(araw,0,13,1);
b = slice(braw,0,13,1);

t00 = user_time();

```

```

intermediate(,t00,start utime);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);
}
}

```

Pass / Fail

7. Record the mean time required to (pole-zero) linear filter a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF03XX-8192.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {pole-zero linear filter on size 8192 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,8192,1);
    a = slice(araw,0,13,1);
  }
}

```

```

    b = slice(braw,0,13,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);

}
}

```

Pass / Fail

9. Record the mean time required to (pole-zero) linear filter a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF03XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {pole-zero linear filter on size 16384 sequence}
  -aliases { xraw = chan_01:data; araw = chan_03:data; braw = chan_02:data }
}

```



```
-inputformat ilwd
-returnprotocol file:/mdc/output/LINFILTPERF03XX-65536.ilwd
-returnformat ilwd
-resultname {time}
-resultcomment {pole-zero linear filter on size 65536 sequence}
-aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
-algorithms {
  x = slice(xraw,0,65536,1);
  a = slice(araw,0,13,1);
  b = slice(braw,0,13,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);
  y = linfoilt(b,a,x); y = linfoilt(b,a,x);

  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}
```

Pass / Fail

15. Record the mean time required to (pole-zero) linear filter a data sequence of length 2^{16} .

Pass / Fail

16. ($N = 2^{17}$) Issue the ldas command


```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);

}
}

```

Pass / Fail

29. Record the mean time required to (pole-zero) linear filter a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to (pole-zero) linear filter a data sequence vs. its length.

Pass / Fail**SUMMARY**

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT**Pass / Fail**

Test Case: LINFILTPERF04

Purpose: Generate performance statistics for the `linfilt` action as a function of the FIR filter order.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. (Order 5 FIR linear filter) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF04XX-5.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {Order 5 FIR linear filter}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);
    a = slice(araw,0,1,1);
    b = slice(braw,0,5,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
  }
}
```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);

}
}

```

Pass / Fail

3. Record the mean time required to apply an order 5 FIR linear filter to a data sequence of length 2^{20} .
Pass / Fail

4. (Order 10 FIR linear filter) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF04XX-10.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {Order 10 FIR linear filter}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);
    a = slice(araw,0,1,1);
    b = slice(brow,0,10,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);
    y = linfoilt(b,a,x); y = linfoilt(b,a,x);

```

```

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(, t32, end utime);
t32 = sub(t32, t00);
intermediate(, t32, raw difference);
time = div(t32, 32);
}
}

```

Pass / Fail

5. Record the mean time required to apply an order 10 FIR linear filter to a data sequence of length 2^{20} .

Pass / Fail

6. (Order 15 FIR linear filter) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF04XX-15.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {Order 15 FIR linear filter}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);
    a = slice(araw,0,1,1);
    b = slice(brow,0,15,1);

    t00 = user_time();

```

```

intermediate(,t00,start utime);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);
y = linfoilt(b,a,x); y = linfoilt(b,a,x);

t32 = user_time();
intermediate(,t32,end utime);
t32 = sub(t32,t00);
intermediate(,t32,raw difference);
time = div(t32,32);
}
}

```

Pass / Fail

7. Record the mean time required to apply an order 15 FIR linear filter to a data sequence of length 2^{20} .
Pass / Fail

8. (Order 20 FIR linear filter) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF04XX-20.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {Order 20 FIR linear filter}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);
    a = slice(araw,0,1,1);
  }
}

```

```

    b = slice(braw,0,20,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);

}
}

```

Pass / Fail

9. Record the mean time required to apply an order 20 FIR linear filter to a data sequence of length 2^{20} .

Pass / Fail

10. (Order 25 FIR linear filter) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF04XX-25.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {Order 25 FIR linear filter}
  -aliases { xraw = chan_01:data; araw = chan_03:data; braw = chan_02:data }
}

```



```

-resultname {time}
-resultcomment {Order 30 FIR linear filter}
-aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
-algorithms {
  x = slice(xraw,0,1048576,1);
  a = slice(araw,0,1,1);
  b = slice(braw,0,30,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}

```

Pass / Fail

13. Record the mean time required to apply an order 30 FIR linear filter to a data sequence of length 2^{20} .

Pass / Fail

14. (Order 35 FIR linear filter) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
}

```

```

-inputformat ilwd
-returnprotocol file:/mdc/output/LINFILTPERF04XX-35.ilwd
-returnformat ilwd
-resultname {time}
-resultcomment {Order 35 FIR linear filter}
-aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
-algorithms {
  x = slice(xraw,0,1048576,1);
  a = slice(araw,0,1,1);
  b = slice(braw,0,35,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);
  y = linfilt(b,a,x); y = linfilt(b,a,x);

  t32 = user_time();
  intermediate(,t32,end utime);
  t32 = sub(t32,t00);
  intermediate(,t32,raw difference);
  time = div(t32,32);
}
}

```

Pass / Fail

15. Record the mean time required to apply an order 35 FIR linear filter to a data sequence of length 2^{20} .

Pass / Fail

16. (Order 40 FIR linear filter) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/LINFILTPERF04XX-40.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {Order 40 FIR linear filter}
  -aliases { xraw = chan_01:data; araw = chan_03:data; brow = chan_02:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);
    a = slice(araw,0,1,1);
    b = slice(braw,0,40,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);
    y = linfilt(b,a,x); y = linfilt(b,a,x);

    t32 = user_time();
    intermediate(,t32,end utime);
    t32 = sub(t32,t00);
    intermediate(,t32,raw difference);
    time = div(t32,32);

  }
}

```

Pass / Fail

17. Record the mean time required to apply an order 40 FIR linear filter to a data sequence of length 2^{20} .

}
}

Pass / Fail

21. Record the mean time required to apply an order 50 FIR linear filter to a data sequence of length 2^{20} .

Pass / Fail

22. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to apply an FIR filter vs. the filter order.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

Test Case: RESAMPLEPERF01

Purpose: Generate performance statistics for `resample` (down-sample by 2 as a function of sequence length).

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($N = 2^{10}$) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-1024.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1024 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1024,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
```

```

        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

3. Record the mean time required to downsample by 2 a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2048 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2048,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

5. Record the mean time required to downsample by 2 a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-4096.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 4096 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,4096,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

7. Record the mean time required to downsample by 2 a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-8192.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 8192 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,8192,1);

    t00 = user_time();

```

```

        intermediate(,t00,start utime);
        y = resample(1,2,x);
        y = resample(1,2,x);
        y = resample(1,2,x);
        y = resample(1,2,x);
        t04 = user_time();
        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

9. Record the mean time required to downsample by 2 a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 16384 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,16384,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

11. Record the mean time required to downsample by 2 a data sequence of length 2^{14} .

Pass / Fail

12. ($N = 2^{15}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-32768.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 32768 downsample by 2}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,32768,1);

      t00 = user_time();
      intermediate(,t00,start utime);
      y = resample(1,2,x);
      y = resample(1,2,x);
      y = resample(1,2,x);
      y = resample(1,2,x);
      t04 = user_time();
      intermediate(,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(,t04,raw difference);
      time = div(t04,4);

    }
  }
}
```

Pass / Fail

13. Record the mean time required to downsample by 2 a data sequence of length 2^{15} .

Pass / Fail

14. ($N = 2^{16}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-65536.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 65536 downsample by 2}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,65536,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(1,2,x);
  y = resample(1,2,x);
  y = resample(1,2,x);
  y = resample(1,2,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

15. Record the mean time required to downsample by 2 a data sequence of length 2^{16} .

Pass / Fail

16. ($N = 2^{17}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-131072.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 131072 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,131072,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
  }
}

```

```

        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

17. Record the mean time required to downsample by 2 a data sequence of length 2^{17} .

Pass / Fail

18. ($N = 2^{18}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-262144.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 262144 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,262144,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

19. Record the mean time required to downsample by 2 a data sequence of length 2^{18} .

Pass / Fail

20. ($N = 2^{19}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-524288.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 524288 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,524288,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

21. Record the mean time required to downsample by 2 a data sequence of length 2^{19} .

Pass / Fail

22. ($N = 2^{20}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-1048576.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1048576 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);

    t00 = user_time();

```

```

    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
}
}

```

Pass / Fail

23. Record the mean time required to downsample by 2 a data sequence of length 2^{20} .

Pass / Fail

24. ($N = 2^{21}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-2097152.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2097152 downsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2097152,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    y = resample(1,2,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
}
}

```

Pass / Fail

25. Record the mean time required to downsample by 2 a data sequence of length 2^{21} .

Pass / Fail

26. ($N = 2^{22}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-4194304.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 4194304 downsample by 2}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,4194304,1);

      t00 = user_time();
      intermediate(,t00,start utime);
      y = resample(1,2,x);
      y = resample(1,2,x);
      y = resample(1,2,x);
      y = resample(1,2,x);
      t04 = user_time();
      intermediate(,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(,t04,raw difference);
      time = div(t04,4);
    }
  }
}
```

Pass / Fail

27. Record the mean time required to downsample by 2 a data sequence of length 2^{22} .

Pass / Fail

28. ($N = 2^{23}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF01XX-8388608.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 8388608 downsample by 2}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,8388608,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(1,2,x);
  y = resample(1,2,x);
  y = resample(1,2,x);
  y = resample(1,2,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

29. Record the mean time required to downsample by 2 a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to downsample by 2 a data sequence vs. its length.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

 New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: RESAMPLEPERF02

Purpose: Generate performance statistics for `resample` (down-sample by 8 as a function of sequence length).

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($N = 2^{10}$) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-1024.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1024 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1024,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
```

```

        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

3. Record the mean time required to downsample by 8 a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2048 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2048,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

5. Record the mean time required to downsample by 8 a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-4096.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 4096 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,4096,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

7. Record the mean time required to downsample by 8 a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-8192.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 8192 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,8192,1);

    t00 = user_time();

```

```

    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);

}
}

```

Pass / Fail

9. Record the mean time required to downsample by 8 a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 16384 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,16384,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);

}
}

```

Pass / Fail

11. Record the mean time required to downsample by 8 a data sequence of length 2^{14} .

Pass / Fail

12. ($N = 2^{15}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-32768.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 32768 downsample by 8}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,32768,1);

      t00 = user_time();
      intermediate(,t00,start utime);
      y = resample(1,8,x);
      y = resample(1,8,x);
      y = resample(1,8,x);
      y = resample(1,8,x);
      t04 = user_time();
      intermediate(,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(,t04,raw difference);
      time = div(t04,4);

    }
  }
}
```

Pass / Fail

13. Record the mean time required to downsample by 8 a data sequence of length 2^{15} .

Pass / Fail

14. ($N = 2^{16}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-65536.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 65536 downsample by 8}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,65536,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(1,8,x);
  y = resample(1,8,x);
  y = resample(1,8,x);
  y = resample(1,8,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

15. Record the mean time required to downsample by 8 a data sequence of length 2^{16} .

Pass / Fail

16. ($N = 2^{17}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-131072.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 131072 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,131072,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
  }
}

```

```

        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

17. Record the mean time required to downsample by 8 a data sequence of length 2^{17} .

Pass / Fail

18. ($N = 2^{18}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-262144.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 262144 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,262144,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

19. Record the mean time required to downsample by 8 a data sequence of length 2^{18} .

Pass / Fail

20. ($N = 2^{19}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-524288.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 524288 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,524288,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

21. Record the mean time required to downsample by 8 a data sequence of length 2^{19} .

Pass / Fail

22. ($N = 2^{20}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-1048576.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1048576 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);

    t00 = user_time();

```

```

        intermediate(,t00,start utime);
        y = resample(1,8,x);
        y = resample(1,8,x);
        y = resample(1,8,x);
        y = resample(1,8,x);
        t04 = user_time();
        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

23. Record the mean time required to downsample by 8 a data sequence of length 2^{20} .

Pass / Fail

24. ($N = 2^{21}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-2097152.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2097152 downsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2097152,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    y = resample(1,8,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

25. Record the mean time required to downsample by 8 a data sequence of length 2^{21} .

Pass / Fail

26. ($N = 2^{22}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-4194304.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 4194304 downsample by 8}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,4194304,1);

      t00 = user_time();
      intermediate(.,t00,start utime);
      y = resample(1,8,x);
      y = resample(1,8,x);
      y = resample(1,8,x);
      y = resample(1,8,x);
      t04 = user_time();
      intermediate(.,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(.,t04,raw difference);
      time = div(t04,4);
    }
  }
}
```

Pass / Fail

27. Record the mean time required to downsample by 8 a data sequence of length 2^{22} .

Pass / Fail

28. ($N = 2^{23}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF02XX-8388608.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 8388608 downsample by 8}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,8388608,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(1,8,x);
  y = resample(1,8,x);
  y = resample(1,8,x);
  y = resample(1,8,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

29. Record the mean time required to downsample by 8 a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to downsample by 8 a data sequence vs. its length.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

 New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: RESAMPLEPERF03

Purpose: Generate performance statistics for `resample` (upsample by 2 as a function of sequence length).

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($N = 2^{10}$) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-1024.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1024 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1024,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
```

```

        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

3. Record the mean time required to upsample by 2 a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2048 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2048,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

5. Record the mean time required to upsample by 2 a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-4096.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 4096 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,4096,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

7. Record the mean time required to upsample by 2 a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-8192.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 8192 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,8192,1);

    t00 = user_time();

```

```

    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
}
}

```

Pass / Fail

9. Record the mean time required to upsample by 2 a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 16384 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,16384,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
}
}

```

Pass / Fail

11. Record the mean time required to upsample by 2 a data sequence of length 2^{14} .

Pass / Fail

12. ($N = 2^{15}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-32768.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 32768 upsample by 2}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,32768,1);

      t00 = user_time();
      intermediate(,t00,start utime);
      y = resample(2,1,x);
      y = resample(2,1,x);
      y = resample(2,1,x);
      y = resample(2,1,x);
      t04 = user_time();
      intermediate(,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(,t04,raw difference);
      time = div(t04,4);

    }
  }
}
```

Pass / Fail

13. Record the mean time required to upsample by 2 a data sequence of length 2^{15} .

Pass / Fail

14. ($N = 2^{16}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-65536.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 65536 upsample by 2}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,65536,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(2,1,x);
  y = resample(2,1,x);
  y = resample(2,1,x);
  y = resample(2,1,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

15. Record the mean time required to upsample by 2 a data sequence of length 2^{16} .

Pass / Fail

16. ($N = 2^{17}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-131072.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 131072 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,131072,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
  }
}

```

```

        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

17. Record the mean time required to upsample by 2 a data sequence of length 2^{17} .

Pass / Fail

18. ($N = 2^{18}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-262144.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 262144 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,262144,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

19. Record the mean time required to upsample by 2 a data sequence of length 2^{18} .

Pass / Fail

20. ($N = 2^{19}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-524288.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 524288 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,524288,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

21. Record the mean time required to upsample by 2 a data sequence of length 2^{19} .

Pass / Fail

22. ($N = 2^{20}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-1048576.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1048576 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);

    t00 = user_time();

```

```

    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
}
}

```

Pass / Fail

23. Record the mean time required to upsample by 2 a data sequence of length 2^{20} .

Pass / Fail

24. ($N = 2^{21}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-2097152.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2097152 upsample by 2}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2097152,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    y = resample(2,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
}
}

```

Pass / Fail

25. Record the mean time required to upsample by 2 a data sequence of length 2^{21} .

Pass / Fail

26. ($N = 2^{22}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-4194304.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 4194304 upsample by 2}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,4194304,1);

      t00 = user_time();
      intermediate(,t00,start utime);
      y = resample(2,1,x);
      y = resample(2,1,x);
      y = resample(2,1,x);
      y = resample(2,1,x);
      t04 = user_time();
      intermediate(,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(,t04,raw difference);
      time = div(t04,4);

    }
  }
}
```

Pass / Fail

27. Record the mean time required to upsample by 2 a data sequence of length 2^{22} .

Pass / Fail

28. ($N = 2^{23}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF03XX-8388608.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 8388608 upsample by 2}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,8388608,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(2,1,x);
  y = resample(2,1,x);
  y = resample(2,1,x);
  y = resample(2,1,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

29. Record the mean time required to upsample by 2 a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to upsample by 2 a data sequence vs. its length.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

 New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: RESAMPLEPERF04

Purpose: Generate performance statistics for `resample` (upsample by 8 as a function of sequence length).

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.

R4/ R8/ C8/ C16

2. ($N = 2^{10}$) Issue the `ldas` command

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-1024.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1024 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1024,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
```

```

        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

3. Record the mean time required to upsample by 8 a data sequence of length 2^{10} .

Pass / Fail

4. ($N = 2^{11}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-2048.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2048 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2048,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

5. Record the mean time required to upsample by 8 a data sequence of length 2^{11} .

Pass / Fail

6. ($N = 2^{12}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-4096.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 4096 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,4096,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

7. Record the mean time required to upsample by 8 a data sequence of length 2^{12} .

Pass / Fail

8. ($N = 2^{13}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-8192.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 8192 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,8192,1);

    t00 = user_time();

```

```

        intermediate(,t00,start utime);
        y = resample(8,1,x);
        y = resample(8,1,x);
        y = resample(8,1,x);
        y = resample(8,1,x);
        t04 = user_time();
        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

9. Record the mean time required to upsample by 8 a data sequence of length 2^{13} .

Pass / Fail

10. ($N = 2^{14}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-16384.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 16384 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,16384,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

11. Record the mean time required to upsample by 8 a data sequence of length 2^{14} .

Pass / Fail

12. ($N = 2^{15}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-32768.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 32768 upsample by 8}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,32768,1);

      t00 = user_time();
      intermediate(.,t00,start utime);
      y = resample(8,1,x);
      y = resample(8,1,x);
      y = resample(8,1,x);
      y = resample(8,1,x);
      t04 = user_time();
      intermediate(.,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(.,t04,raw difference);
      time = div(t04,4);

    }
  }
}
```

Pass / Fail

13. Record the mean time required to upsample by 8 a data sequence of length 2^{15} .

Pass / Fail

14. ($N = 2^{16}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-65536.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 65536 upsample by 8}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,65536,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(8,1,x);
  y = resample(8,1,x);
  y = resample(8,1,x);
  y = resample(8,1,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

15. Record the mean time required to upsample by 8 a data sequence of length 2^{16} .

Pass / Fail

16. ($N = 2^{17}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-131072.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 131072 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,131072,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
  }
}

```

```

        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

17. Record the mean time required to upsample by 8 a data sequence of length 2^{17} .

Pass / Fail

18. ($N = 2^{18}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-262144.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 262144 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,262144,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

19. Record the mean time required to upsample by 8 a data sequence of length 2^{18} .

Pass / Fail

20. ($N = 2^{19}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-524288.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 524288 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,524288,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

21. Record the mean time required to upsample by 8 a data sequence of length 2^{19} .

Pass / Fail

22. ($N = 2^{20}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-1048576.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 1048576 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,1048576,1);

    t00 = user_time();

```

```

        intermediate(,t00,start utime);
        y = resample(8,1,x);
        y = resample(8,1,x);
        y = resample(8,1,x);
        y = resample(8,1,x);
        t04 = user_time();
        intermediate(,t04,end utime);
        t04 = sub(t04,t00);
        intermediate(,t04,raw difference);
        time = div(t04,4);
    }
}

```

Pass / Fail

23. Record the mean time required to upsample by 8 a data sequence of length 2^{20} .

Pass / Fail

24. ($N = 2^{21}$) Issue the ldas command

```

ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388603XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-2097152.ilwd
  -returnformat ilwd
  -resultname {time}
  -resultcomment {time for size 2097152 upsample by 8}
  -aliases { xraw = chan_01:data }
  -algorithms {
    x = slice(xraw,0,2097152,1);

    t00 = user_time();
    intermediate(,t00,start utime);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    y = resample(8,1,x);
    t04 = user_time();
    intermediate(,t04,end utime);
    t04 = sub(t04,t00);
    intermediate(,t04,raw difference);
    time = div(t04,4);
  }
}

```

Pass / Fail

25. Record the mean time required to upsample by 8 a data sequence of length 2^{21} .

Pass / Fail

26. ($N = 2^{22}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-4194304.ilwd
    -returnformat ilwd
    -resultname {time}
    -resultcomment {time for size 4194304 upsample by 8}
    -aliases { xraw = chan_01:data }
    -algorithms {
      x = slice(xraw,0,4194304,1);

      t00 = user_time();
      intermediate(,t00,start utime);
      y = resample(8,1,x);
      y = resample(8,1,x);
      y = resample(8,1,x);
      y = resample(8,1,x);
      t04 = user_time();
      intermediate(,t04,end utime);
      t04 = sub(t04,t00);
      intermediate(,t04,raw difference);
      time = div(t04,4);
    }
  }
}
```

Pass / Fail

27. Record the mean time required to upsample by 8 a data sequence of length 2^{22} .

Pass / Fail

28. ($N = 2^{23}$) Issue the ldas command

```
ldasJob { -name user -password ***** -email user@host }
  { conditionData
    -inputprotocol file:/mdc/input/N8388608XX.sim
    -inputformat ilwd
    -returnprotocol file:/mdc/output/RESAMPLEPERF04XX-8388608.ilwd
    -returnformat ilwd
```

```

-resultname {time}
-resultcomment {time for size 8388608 upsample by 8}
-aliases { xraw = chan_01:data }
-algorithms {
  x = slice(xraw,0,8388608,1);

  t00 = user_time();
  intermediate(,t00,start utime);
  y = resample(8,1,x);
  y = resample(8,1,x);
  y = resample(8,1,x);
  y = resample(8,1,x);
  t04 = user_time();
  intermediate(,t04,end utime);
  t04 = sub(t04,t00);
  intermediate(,t04,raw difference);
  time = div(t04,4);
}
}

```

Pass / Fail

29. Record the mean time required to upsample by 8 a data sequence of length 2^{23} .

Pass / Fail

30. Create, using the data recorded here and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to upsample by 8 a data sequence vs. its length.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Test Case: FFTPERF01
Purpose: Generate performance statistics for FFT action.
Tester: _____
Test machine: _____
Date (mm/dd/yy): ____/____/____ **Time:** _____

NOTE: This test should be performed for real float (R4), real double (R8), complex float (C8), and complex double (C16) data types.

ENVIRONMENT AND PREREQUISITES

The following input data sets are required for this test:

1. N8388608XX.sim

where XX is either R4, R8, C8, or C16.

PROCEDURE

1. Indicate which data type is currently being tested.
2. For each length N in Tables 1-5, issue the `ldas` command

R4/ R8/ C8/ C16

```
ldasJob { -name user -password ***** -email user@host }
{ conditionData
  -inputprotocol file:/mdc/input/N8388608XX.sim
  -inputformat ilwd
  -returnprotocol file:/mdc/output/FFTPERF01XX-N.ilwd
  -returnformat ilwd
  -resultname {None}
  -resultcomment {None}
  -aliases { data = chan_01:data }
  -algorithms {
    x = slice(data, 0, N, 1);
    fft(x);
    ti = user_time();
    fft(x); fft(x); fft(x); fft(x); fft(x);
    fft(x); fft(x); fft(x); fft(x); fft(x);
    fft(x); fft(x); fft(x); fft(x); fft(x);
    fft(x); fft(x); fft(x); fft(x); fft(x);
    tf = user_time();
    t = sub(tf, ti);
    div(t, 20.0);
    intermediate(, , time, {Time to fft data for length N});"
  }
}
```

}

Pass / Fail

3. In Tables 1-5, record the mean time required to FFT a data sequence of length N .

Pass / Fail

4. Create, using the data recorded Tables 1-5, and any convenient plotting program (*e.g.*, excel), a figure showing the mean time to FFT a data sequence vs. its length.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

N (length of data)	time required for FFT (s)	
1024		Pass / Fail
2048		Pass / Fail
3072		Pass / Fail
4094		Pass / Fail
5120		Pass / Fail
6144		Pass / Fail
8192		Pass / Fail
9216		Pass / Fail
10240		Pass / Fail
12288		Pass / Fail
15360		Pass / Fail
16384		Pass / Fail
18432		Pass / Fail
20480		Pass / Fail
24576		Pass / Fail
25600		Pass / Fail
27648		Pass / Fail
30720		Pass / Fail
32768		Pass / Fail
36864		Pass / Fail
40960		Pass / Fail
46080		Pass / Fail
49152		Pass / Fail
51200		Pass / Fail
55296		Pass / Fail
61440		Pass / Fail
65536		Pass / Fail
73728		Pass / Fail
76800		Pass / Fail
81920		Pass / Fail
82944		Pass / Fail
92160		Pass / Fail
98304		Pass / Fail
102400		Pass / Fail
110592		Pass / Fail
122880		Pass / Fail
128000		Pass / Fail
131072		Pass / Fail
138240		Pass / Fail
147456		Pass / Fail

Table 1: Time required to FFT a data sequence as a function of its length.

N (length of data)	time required for FFT (s)	
153600		Pass / Fail
163840		Pass / Fail
165888		Pass / Fail
184320		Pass / Fail
196608		Pass / Fail
204800		Pass / Fail
221184		Pass / Fail
230400		Pass / Fail
245760		Pass / Fail
248832		Pass / Fail
256000		Pass / Fail
262144		Pass / Fail
276480		Pass / Fail
294912		Pass / Fail
307200		Pass / Fail
327680		Pass / Fail
331776		Pass / Fail
368640		Pass / Fail
384000		Pass / Fail
393216		Pass / Fail
409600		Pass / Fail
414720		Pass / Fail
442368		Pass / Fail
460800		Pass / Fail
491520		Pass / Fail
497664		Pass / Fail
512000		Pass / Fail
524288		Pass / Fail
552960		Pass / Fail
589824		Pass / Fail
614400		Pass / Fail
640000		Pass / Fail
655360		Pass / Fail
663552		Pass / Fail
691200		Pass / Fail
737280		Pass / Fail
746496		Pass / Fail
768000		Pass / Fail
786432		Pass / Fail
819200		Pass / Fail

Table 2: Time required to FFT a data sequence as a function of its length (continued).

N (length of data)	time required for FFT (s)	
829440		Pass / Fail
884736		Pass / Fail
921600		Pass / Fail
983040		Pass / Fail
995328		Pass / Fail
1024000		Pass / Fail
1048576		Pass / Fail
1105920		Pass / Fail
1152000		Pass / Fail
1179648		Pass / Fail
1228800		Pass / Fail
1244160		Pass / Fail
1280000		Pass / Fail
1310720		Pass / Fail
1327104		Pass / Fail
1382400		Pass / Fail
1474560		Pass / Fail
1492992		Pass / Fail
1536000		Pass / Fail
1572864		Pass / Fail
1638400		Pass / Fail
1658880		Pass / Fail
1769472		Pass / Fail
1843200		Pass / Fail
1920000		Pass / Fail
1966080		Pass / Fail
1990656		Pass / Fail
2048000		Pass / Fail
2073600		Pass / Fail
2097152		Pass / Fail
2211840		Pass / Fail
2239488		Pass / Fail
2304000		Pass / Fail
2359296		Pass / Fail
2457600		Pass / Fail
2488320		Pass / Fail
2560000		Pass / Fail
2621440		Pass / Fail
2654208		Pass / Fail
2764800		Pass / Fail

Table 3: Time required to FFT a data sequence as a function of its length (continued).

N (length of data)	time required for FFT (s)	
2949120		Pass / Fail
2985984		Pass / Fail
3072000		Pass / Fail
3145728		Pass / Fail
3200000		Pass / Fail
3276800		Pass / Fail
3317760		Pass / Fail
3456000		Pass / Fail
3538944		Pass / Fail
3686400		Pass / Fail
3732480		Pass / Fail
3840000		Pass / Fail
3932160		Pass / Fail
3981312		Pass / Fail
4096000		Pass / Fail
4147200		Pass / Fail
4194304		Pass / Fail
4423680		Pass / Fail
4478976		Pass / Fail
4608000		Pass / Fail
4718592		Pass / Fail
4915200		Pass / Fail
4976640		Pass / Fail
5120000		Pass / Fail
5242880		Pass / Fail
5308416		Pass / Fail
5529600		Pass / Fail
5760000		Pass / Fail
5898240		Pass / Fail
5971968		Pass / Fail
6144000		Pass / Fail
6220800		Pass / Fail
6291456		Pass / Fail
6400000		Pass / Fail
6553600		Pass / Fail
6635520		Pass / Fail
6912000		Pass / Fail
7077888		Pass / Fail
7372800		Pass / Fail
7464960		Pass / Fail

Table 4: Time required to FFT a data sequence as a function of its length (continued).

N (length of data)	time required for FFT (s)	
7680000		Pass / Fail
7864320		Pass / Fail
7962624		Pass / Fail
8192000		Pass / Fail
8294400		Pass / Fail
8388608		Pass / Fail
7464960		Pass / Fail
7680000		Pass / Fail
7864320		Pass / Fail
7962624		Pass / Fail
8192000		Pass / Fail
8294400		Pass / Fail
8388608		Pass / Fail

Table 5: Time required to FFT a data sequence as a function of its length (continued).

Test Case: PIPE01
Purpose: Test functioning of a complex pipeline
Tester: _____
Test machine: _____
Date (mm/dd/yy): ___/___/___ **Time:** _____

ENVIRONMENT AND PREREQUISITES The following input data sets are required for this test:

1. ...

PROCEDURE

1. Issue the following LDAS command; *following completion*, repeat with an incremented input and output file until all 40 input files have been processed:

```

ldasJob { -name dcapi -password ***** \
          -email dcapi@ligo.caltech.edu } \
{ conditionData -inputprotocol file:/[[INDATA]] -inputformat ilwd \
  -returnprotocol file:/[[OUTDATA]] -returnformat ilwd \
  -resultcomment {[[COMMENT]]} -resultname {[[RSLTNME]] } \
  -aliases {y10 = [[CHAN]]} \
  -algorithms { \
    y10 = slice(y10,0,4194304,1); \
    y10 = resample(1,4,data); \
    y09 = resample(1,2,y10); \
    y08 = resample(1,2,y09); \
  }

```

```
y07 = resample(1,2,y08); \  
y10 = mix(0.5,0.0,y10); \  
y09 = mix(0.5,0.0,y09); \  
y08 = mix(0.5,0.0,y08); \  
y07 = mix(0.5,0.0,y07); \  
y10 = resample(1,4,y10); \  
y09 = resample(1,4,y09); \  
y08 = resample(1,4,y08); \  
y07 = resample(1,4,y07); \  
tmp = real(y10); \  
sr10 = all(tmp); \  
tmp = imag(y10); \  
si10 = all(tmp); \  
tmp = real(y09); \  
sr09 = all(tmp); \  
tmp = imag(y09); \  
si09 = all(tmp); \  
tmp = real(y08); \  
sr08 = all(tmp); \  
tmp = imag(y08); \  
si08 = all(tmp); \  
tmp = real(y07); \  
sr07 = all(tmp); \  
tmp = imag(y07); \  
si07 = all(tmp); \  
p10 = psd(y10,f10); \  
intermediate(,"p10","512 Hz about 1024 Hz"); \  
value(f10); \  
intermediate(,"f10","freq/256 Hz"); \  
p09 = psd(y09,f09); \  
intermediate(,"p09","256 Hz about 512 Hz"); \  
value(f09); \  
intermediate(,"f09","freq/128 Hz"); \  
p08 = psd(y08,f08); \  
intermediate(,"p08","128 Hz about 256 Hz"); \  
value(f08); \  
intermediate(,"f08","freq/64 Hz"); \  
p07 = psd(y07,f07); \  
intermediate(,"p07","64 Hz about 128 Hz"); \  
value(f07); \  
intermediate(,"f07","freq/32 Hz"); \  
} \  
}
```

Pass / Fail

2. Move the 40 output files to MDC/Pipeline.

Pass / Fail

3. Inspect the statistics and the power spectral densities recorded in the output files for correctness.

Pass / Fail

SUMMARY

Known faults encountered – list bug IDs:

New faults submitted – list bug IDs:

TEST RESULT

Pass / Fail

Draft

Test Case: PIPE02

Purpose: Test the data conditioning API's ability to execute multiple chains simultaneously.

Tester: _____

Test machine: _____

Date (mm/dd/yy): ____/____/____ **Time:** _____

ENVIRONMENT AND PREREQUISITES The following input data sets are required for this test:

1. Data files that contain a single channel with 2^{20} points.

PROCEDURE

1. Issue 8 copies of the LDAS command

```
ldasJob {-name dcapi -password ***** \
        -email dcapi@ligo.caltech.edu } \
{ conditionMdcData -inputprotocol file:/mdc/input/[[INDATA]] \
  -returnprotocol file:/mdc/output/mdc_data.txt -returnformat ilwd \
  -resultcomment {'Output data'} -resultname {0Data} \
  -aliases { data = [[CHAN]] } \
  -algorithms { \
    data = value(data);
    intermediate(, 'Input data', 'IData'); \
    y = fft(data); \
    x = ifft(data); \
    x - data; \
  } \
}
```

in succession, and then a single copy of the command for each that finishes until 32 commands have completed. **Pass / Fail**

2. Verify that eight commands executed simultaneously. **Pass / Fail**
3. Verify that the results from the first and last command are correct to numerical precision. **Pass / Fail**

SUMMARY

Known faults encountered – list bug IDs: _____

New faults submitted – list bug IDs: _____

TEST RESULT

Pass / Fail

Draft

C Final Report

Draft