SCUBA-2 FTS Project Office
University of Lethbridge
Physics Department
4401 University Drive
Lethbridge, Alberta
CANADA
T1K 3M4

Tel: 1-403-329-2771
Fax: 1-403-329-2057
Email: brad.gom@uleth.ca
WWW: http://research.uleth.ca/scuba2/

Document Title: **FTS-2 Data Reduction Engine**

Document Number: **SC2/FTS/SOF/001**

Issue: **Version 2.2**

Date: **2 November 2006**

Document Prepared By:	Baoshe Zhang FTS-2 SW Engineer	Signature and Date:	02/11/06
Document Approved By:	B. G. Gom FTS-2 Project Manager	Signature and Date:	David A Naylor 02/11/06
Document Released By:	J. Molnar Canadian Project Manager	Signature and Date:	02/11/06

Change Record

Issue	Date	Section(s) Affected	Description of Change / Change Request Reference / Remarks
0.1	17/05/05	All	First draft
0.2	31/05/05	All	Second draft
1.0	20/06/05	All	PDR version
2.0	22/03/06	All	CDR draft
2.1	1/11/06	All	CDR version, minor updates.
2.2	2/11/06	A.1	Fixed last PCF equation

Contents

Applicable and Referenced Documents

Document Number	*Title*	*Number & Issue*
SC2/SRE/S210/001	Data Reduction Software Requirements Document	1.56
SC2/SOF/S210/001	SCUBA-2 Pipeline Architecture	1.25
SC2/SOF/IC210/01	DA/DR Pipeline Interface Control Document	1.30
HERSCHEL-HSC-DOC-0517	Herschel Interactive Analysis: A Basic User's Manual	0.4
Starlink User Note 251	Getting Started with the Starlink Java Infrastructure and Applications Set	1.0
SC2/SRE/SC200/002	Functional and Performance Requirements for SCUBA-2	
SC2/FTS/SCE/001	FTS-2 Science Case	2.0
SC2/FTS/SRE/001	FTS-2 Functional and Performance Requirements	3.0
SC2/FTS/SOF/002	FTS-2 to OCS ICD	1.0
SC2/FTS/SOF/003	FTS-2 Display System	2.0
SPIRE-UOL-REP-02220	Performance Test Report for the SPIRE Interactive Analysis – WP Fourier Transform	1.0
www.aao.gov.au/drama	DRAMA protocol	
ws.apache.org/axis	Apache Axis	

Summary

The SCUBA-2 data reduction pipeline is designed to call the corresponding algorithm engine for the current instrument when the pipeline detects a new data file either from the Data Acquisition System (on-line mode) or from the data archive (off-line mode). After the algorithm engine completes the relevant tasks or exits, the SCUBA-2 data reduction pipeline will regain the control of the data flow.

The FTS-2 data reduction algorithm engine is written in Java and consists of two major parts: an interface layer and a core layer. The interface layer provides a message interface for the SCUBA-2 data reduction pipeline to invoke the functions of the core layer. The core layer is made up of five independent modules: I/O, Interpolation, Phase Correction, FFT, and the Quick Look (QL) display system.

This document describes the details of the interface layer and core layer of the FTS-2 data reduction algorithm engine.

1. Overview of the SCUBA-2 Data Reduction Pipeline

Figure 1 illustrates the SCUBA-2 Data Reduction Pipeline architecture. The Pipeline itself is written in object-oriented Perl. While the Recipes concentrate on reduction control, the bulk of the algorithmic data processing is performed using separate “algorithm engines”. Since the implementation of the algorithm engines is independent of the Pipeline itself, the FTS-2 algorithm engine can be written in Java to exploit existing Herschel SPIRE code written by members of our group.

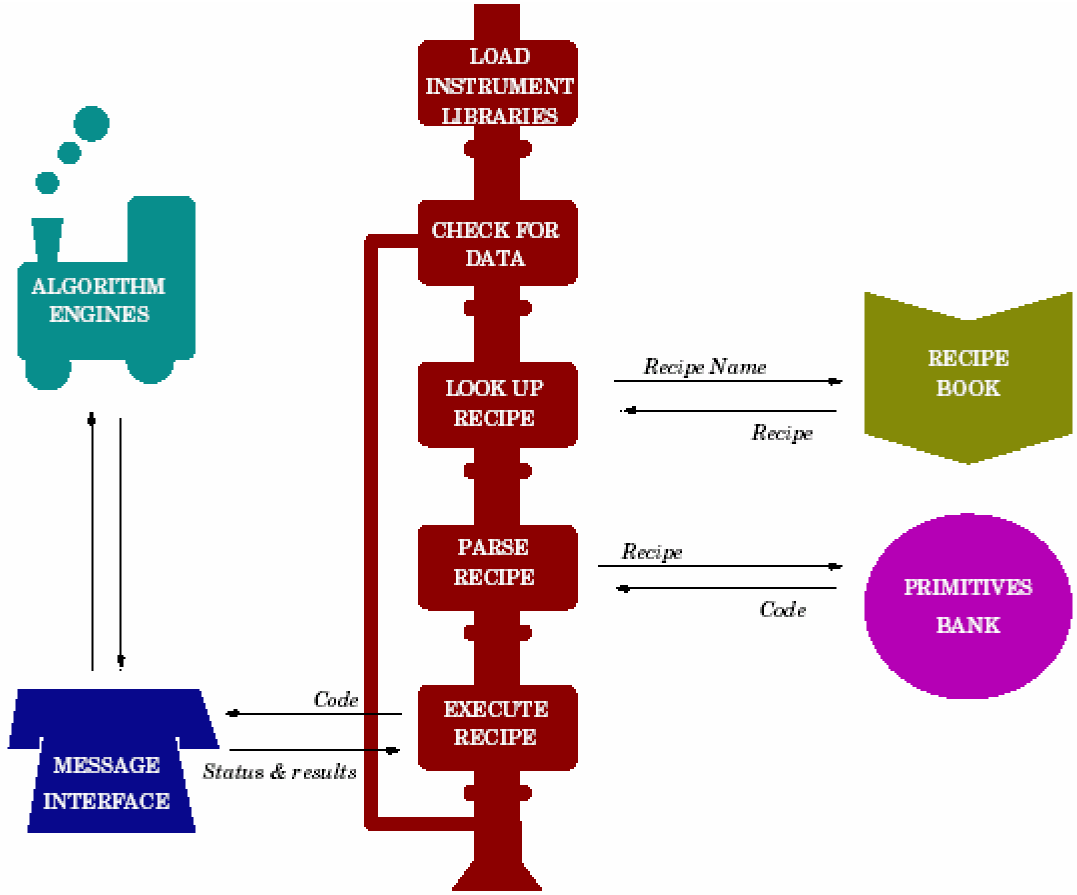

Figure 1. SCUBA-2 DR Pipeline

2. Structure of the FTS-2 Engine

The FTS-2 algorithm engine uses a multi-tier structure consisting of three separate modules (referred to as 'actions' for DRAMA, or 'operations' for SOAP):

Module	Function	DRAMA action	SOAP operation
set_parameters	set the data reduction parameters	SETPARAMETERS	setParameters
data_reduction	perform the core numerical computation of FTS-2 data reduction	DATAREDUCTION	dataReduction
exit	stop FTS-2 Engine and exit	EXIT	exitSOAP

From the perspective of the Pipeline, each action or operation has a corresponding primitive. The Pipeline can call these three actions (or operations) through the FTS-2 DRAMA or SOAP message interface.

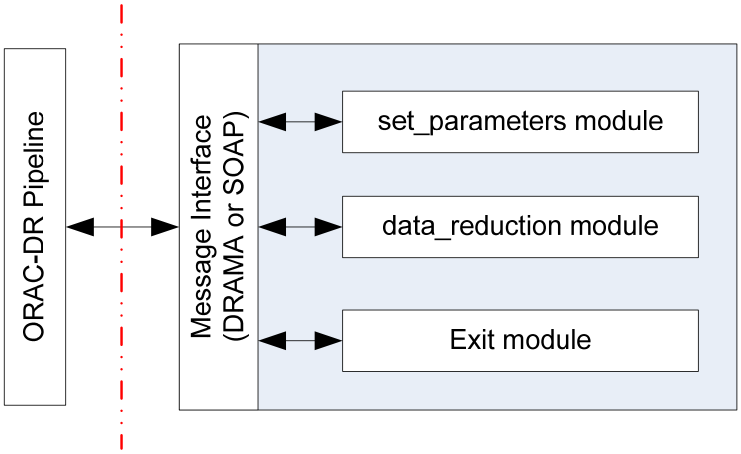

Figure 2. FTS-2 Algorithm Engine messaging overview.

3. Core Layer

In this section, the modules of FTS-2 algorithm engine core layer are described in detail. Figure 3 shows a schematic of the FTS-2 data reduction module:

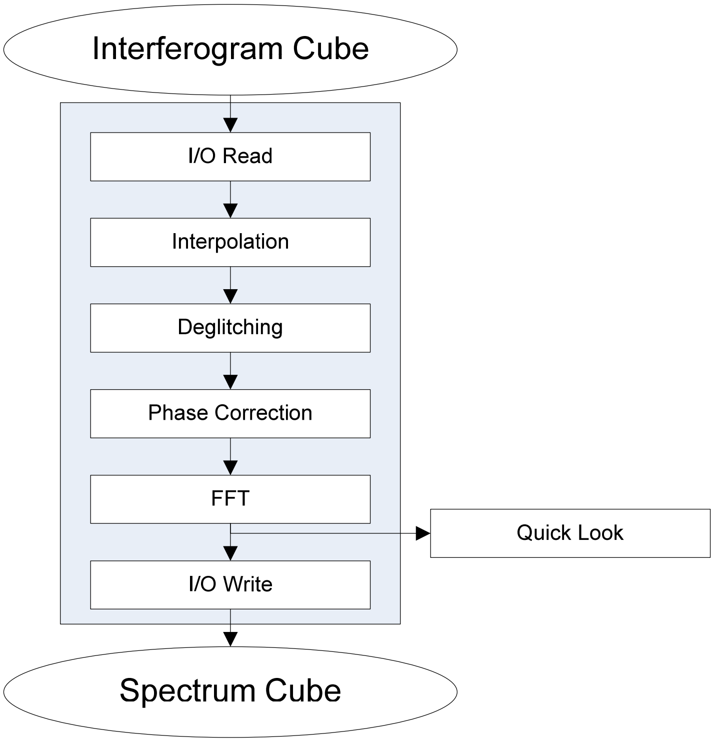

Figure 3. FTS-2 Data Reduction module overview.

In order to separate the core numeric processing from both the data access layer and the message processing layer, the FTS-2 data reduction module is implemented as a Java class: **ca.uol.aig.fts.drpipeline.DRPipeline**. This class integrates I/O, Interpolation, Phase Correction, FFT and Quick Look into a pipeline to fully process FTS-2 interferogram data. Both single-threaded and multi-threaded versions are available:

Single-Thread Version:

```
DRPipeline(String in, String out, int pcfSize_h,
           int dsSize, int ssSize, int fittingDegree,
           double weight_limit, double ZPD_value,
           double wn_lBound, double wn_uBound,
           int deglitching_flag)
```

Multi-Threaded Version:

```
DRPipeline(String in, String out, int pcfSize_h,
           int dsSize, int ssSize, int fittingDegree,
            double weight_limit, double ZPD_value,
            double wn_lBound, double wn_uBound, int deglitching_flag,
            int numThread)
```

If the stage is configured to return position values with 0 occurring exactly at ZPD, then the DRPipeline call can be simplified slightly:

Single-Threaded Version:

```
DRPipeline(String in, String out, int pcfSize_h, int dsSize,
           int ssSize, int fittingDegree, double weight_limit,
           double wn_lBound, double wn_uBound,
           int deglitching_flag)
```

Multi-Threaded Version:

```
DRPipeline(String in, String out, int pcfSize_h,
           int dsSize, int ssSize, int fittingDegree,
           double weight_limit, double wn_lBound,
           double wn_uBound, int deglitching_flag,
           int numThread)
```

The parameters are described in the following table:

In	path of the raw data file
Out	path of the reduced data file
pcfSize_h	half of the length of the phase correction function (points)
dsSize	half of the length of the double-sided interferogram (points)
ssSize	length of the single-sided interferogram (points)
fittingDegree	the polynomial degree of phase fitting function
Weight_limit	Threshold amplitude (expressed as a fraction), above which the points will be taken into account in phase fitting
ZPD_value	The position of ZPD. Usually, this is a measured value.
wn_lBound	The fractional position (0 – 1) in the phase array below which phase data will not be included in the phase fitting. The lowest wavenumber used in phase-fitting is wn_lBound x dsSize x Nyquist.
wn_uBound	The fractional position (0 – 1) in the phase array above which phase data will not be included in the phase fitting. The highest wavenumber used in phase-fitting is wn_uBound x dsSize x Nyquist.
deglitching_flag	Flag for deglitching: When deglitching_flag=1, deglitch the core part of the interferogram. When deglitching_flag=2, deglitch the tail part of the interferogram. When deglitching_flag=3, deglitch the core and tail part of the interferogram. Otherwise, no deglitching.
numThread	The number of computation threads. When numThread ≤ 1, DRPipeline uses the single-thread version. When numThread ≥ the width of the array (For FTS-2, the width of the array is 40), the number of computation threads will be equal to the width of the array.

In order to reduce the FFT computation time, the size of the double-sided and single-sided parts of the interferogram should be factorable as $2^a3^b5^c7^d$ (a, b, c are any integer that is greater than or equal to 0, d is 0, 1). When the values of dsSize and ssSize do not satisfy this requirement, DRPipeline will use new values which are nearest to their input counterparts in DRPipeline and meet this requirement. Appendix B explains the relations between dsSize, ssSize, pcfSize_h, and ZPD_value in details. The width of the array does not need to be divisible by numThread. That is, for FTS-2, numThread does not need to be a factor of 40. DRPipeline knows how to load the data reduction tasks evenly to these numThread threads.

Since the FTS-2 engine may consume a lot of memory, the garbage collection of Java must be made to be as efficient as possible. The garbage collection thread should therefore be integrated into the DRPipeline class.

3.1. I/O

The FTS-2 engine I/O function of is implemented by Java class **ca.uol.aig.fts.io.NDFIO**.

3.2. Interpolation

The FTS-2 engine interpolation function is implemented in the Java class **ca.uol.aig.fts.fitting.CubicSpline**.

3.3. Deglitching

The FTS-2 engine deglitching function is implemented in the Java class **ca.uol.aig.fts.deglitch.Deglitching**.

3.4. Phase Correction

The FTS-2 engine Phase Correction function is implemented in the Java class **ca.uol.aig.fts.phasecorrection.PhaseCorrection**.

3.5. FFT

The FTS-2 engine uses the classes **RealDoubleFFT**, **RealDoubleFFT_Even**, and **RealDoubleFFT_Odd** of the Java package **ca.uol.aig.fftpack** to do all FFT computations.

3.6. Quick Look

The FTS-2 engine Quick Look function is implemented in the Java class **ca.uol.aig.fts.display.QuickLookXY**.

4. Interface Layer

The algorithm engines are started by the Pipeline on demand and controlled via the appropriate messaging system. As stated, the Pipeline itself is flexible enough to support any messaging scheme so long as a single message is sent to the task and a return status made available on completion.

In a client/server scheme, the algorithm engines are the servers that provide data reduction calculation services and the Pipeline is the client that is responsible for initiating the network sessions and passing the parameters to the algorithm engines. Actually, the interface layer is a messaging mechanism between the algorithm engines and the Pipeline.

The FTS-2 Java DR engine provides two independent messaging layers to the Pipeline: DRAMA and SOAP. Either of them can be used for communication between the FTS-2 engine and the Pipeline. Meanwhile, Drama2FTS and SOAP2FTS assume that the ZPD value is 0.

4.1. DRAMA Interface Layer

The DRAMA system was developed by the Anglo-Australian Observatory (AAO) to meet the requirements for a fast, distributed environment. The DJAVA DRAMA package, a Java interface to the AAO's DRAMA API, is used to integrate the DRAMA communication system into the FTS-2 Java DR Engine.

The Java class **ca.uol.aig.fts.message.Drama2FTS** implements a DRAMA task, which is made up of three DRAMA actions: SETPARAMETERS, DATAREDUCTION, and EXIT.

Action SETPARAMETERS sets the value for the following parameters:

- pcfSize_h (default value = 80)
- dsSize (default value = 300)
- ssSize (default value = 300)
- fittingDegree (default value = 2)
- weight_limit (default value = 0.01)
- wn_lBound (default value = 0.05)
- wn_uBound (default value = 1.00)
- deglitch (default value = 0)
- numThread (default value = 1)

Action DATAREDUCTION has two input parameters: in and out. These parameters specify the full path of the raw data file and the processed data file respectively.

Action EXIT will cause this drama task to stop execution and exit.

The following demo Java program, TestDramaServer.java, starts a FTS-2 DR DRAMA task, Drama2FTS.

```
import ca.uol.aig.fts.message.Drama2FTS;

public class TestDramaServer
{
    public static void main(String[] args)
    {
         Drama2FTS df = new Drama2FTS("Drama2FTS");
    }
}
```

The choice of how to invoke the DRAMA actions of this DRAMA task is mainly up to the implementation of the Perl/DRAMA XS interface. Here, the DRAMA program 'ditscmd' is used to demonstrate the way these three DRAMA actions of this DRAMA task are called:

`ditscmd Drama2FTS EXIT`	Perform action EXIT
`ditscmd Drama2FTS SETPARAMETERS "pcfSize_h=60" "dsSize=300" "ssSize=6000" "fittingDegree=2" "weight_limit=0.1" "wn_lBound=0.05" "wn_uBound=1.0" "deglitch=3" "numThread=4"`	Perform action SETPARAMETERS to set pcfSize_h to 60, dsSize to 300, ssSize=6000, fittingDegree to 2, weight_limit = 0.1, wn_lBound = 0.05, wn_uBound = 1.0, deglitch = 3, and numThread = 2.
`Ditscmd Drama2FTS DATAREDUCTION "in=test_rawdata" "out=test_reduced"`	Perform action DATAREDUCTION. The parameter, in, points to the raw input data file, and the parameter, out, points to the processed output data file.

4.2. SOAP Interface Layer

SOAP is an XML-based communication protocol and encoding format for inter-application communication. It is widely viewed as the backbone to a new generation of cross-platform cross-language distributed computing applications, termed Web Services. Apache's Axis (**A**pache E**X**tensible **I**nteraction **S**ystem), a SOAP engine, is used as a framework for constructing SOAP processors such as clients, servers, gateways, etc. The current version of Axis is written in Java and integrated into the StarJava package.

The Java class **ca.uol.aig.fts.message.SOAP2FTS** implements a SOAP web service, which is made up of three SOAP operations: setParameters, dataReduction, and exitSOAP. These are identical to the DRAMA actions described above, however, the major difference between the DRAMA and SOAP implementations is that DRAMA uses named parameters and SOAP (Apache Axis) uses position parameters.

Operation setParameters can set a new value for the following parameters:

- pcfSize_h
- dsSize
- ssSize
- fittingDegree
- weight_limit
- wn_lBound
- wn_uBound
- deglitching_flag
- numThread

Operation dataReduction has two input parameters, which specify the full paths of a raw data file and its processed data file respectively.

Operation exitSOAP will cause this SOAP web service to stop execution and exit.

The following demo Java program, TestSOAPServer.java, starts a FTS-2 DR SOAP web service with the network service port 8082.

```
import ca.uol.aig.fts.message.SOAP2FTS;

public class TestSOAPServer
{
   public static void main(String[] arg)
   {
        String wsddFile = "/DR2FTS/test/test_soap/fts2deploy.wsdd";
        int soapServerPort = 8082;
        SOAP2FTS sf = new SOAP2FTS();
        sf.startSOAPServer(wsddFile, soapServerPort);
   }
}
```

In order to be deployed as a web service, each SOAP web service must have a web service deployment descriptor (WSDD) file. In this demo program, the WSDD file is /DR2FTS/test/fts2deploy.wsdd, which specifies the name of this web service as SOAP2FTS.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
   xmlns="http://xml.apache.org/axis/wsdd/"
   xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <service name="SOAP2FTS" provider="Handler" use="literal" style="rpc">
      <parameter name="className" value="ca.uol.aig.fts.message.SOAP2FTS"/>
      <parameter name="allowedMethods" value="*"/>
      <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
```

```
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>
</deployment>
```

Although this SOAP web service can be invoked in many different ways, the following Java program, TestSOAPClient, is only used to illustrate how to invoke three operations of this web service by Apache Axis itself.

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class TestSOAPClient
{
    public static void main(String[] args)
    {
        try
        {
            String inPath = "/DR2FTS/test/test_soap/dr/";
            String outPath = "/DR2FTS/test/test_soap/dr/";

            /* set the network address of the SOAP Server */
            String endpoint = "http://localhost:8082/services/SOAP2FTS";

            Service  service = new Service();
            Call call = (Call)service.createCall();

            call.setTargetEndpointAddress( new java.net.URL(endpoint) );

            /* call the operation: setParameters */
            call.setOperationName(new QName("http://www.uleth.ca/", "setParameters"));
            int pcfSize_h = 60;
            int dsSize = 300;
            int ssSize = 6000;
            int fittingDegree = 2;
            float weight_limit = 0.1F;
            double wn_lBound = 0.05;
            double wn_uBound = 0.95;
            int deglitching_flag = 0;
            int numThread = 2;
            call.invoke(new Object[] {pcfSize_h, dsSize, ssSize, fittingDegree,
                                weight_limit, wn_lBound, wn_uBound,
                                deglitching_flag, numThread});

            /* call the operation: dataReduction */
            call.setOperationName(new QName("http://www.uleth.ca/", "dataReduction"));
            String rawDataFile = "abc";
            String reducedDataFile = "abc0";
            call.invoke(new Object[] {inPath + rawDataFile, outPath + reducedDataFile});

            /* call the operation: exitSOAP (stop the SOAP server) */
            call.setOperationName(new QName("http://www.uleth.ca/", "exitSOAP"));
            call.invoke(new Object[] {});
        }
        catch(Exception e)
        {
            System.err.println(e.toString());
        }
    }
}
```

5. Benchmarks

The FTS-2 engine was benchmarked on a Linux Fedora platform with a 2.8Ghz P4 CPU and 1 GB RAM. The benchmarks do not include any time that ORAC-DR would take to call the algorithm engine. In general, the phase correction function (PCF) decays very quickly. Usually, in order to reduce the calculation time, only the points of PCF around 0 are taken into account. Larger PCF lengths work better, but require more computation time. The following tables list three benchmarks with PCF=40, PCF=80, and PCF=160 respectively. There is no need for the PCF size to ever exceed 160.

Table 1. Benchmarks with PCF=40

	Interferogram Length (points)			Benchmarks (s)				Total Time (s)	
Scan Mode	**Short Wing**	**Long Wing**	**Total**	**I/O(R/W)**	**Interpolation**	**Phase Correction**	**FFT**	**Acquisition**	**Processing**
SED 850 Band	180	180	360	0.20/0.14	0.18	0.63	0.047	1.80	1.20
SED Dual Band	300	300	600	0.21/0.18	0.25	0.85	0.063	3.00	1.55
Spectral Line 850 Band	180	3000	3180	0.28/1.07	0.89	2.20	0.408	15.90	4.85
Spectral Line Dual Band	300	5000	5300	0.35/1.81	1.37	3.41	0.681	26.50	7.62

Table 2. Benchmarks with PCF=80

	Interferogram Length (points)			Benchmarks (s)				Total Time (s)	
Scan Mode	**Short Wing**	**Long Wing**	**Total**	**I/O(R/W)**	**Interpolation**	**Phase Correction**	**FFT**	**Acquisition**	**Processing**
SED 850 Band	180	180	360	0.20/0.13	0.18	0.69	0.044	1.80	1.24
SED Dual Band	300	300	600	0.20/0.16	0.25	1.00	0.055	3.00	1.67
Spectral Line 850 Band	180	3000	3180	0.28/1.01	0.90	3.50	0.407	15.90	6.10
Spectral Line Dual Band	300	5000	5300	0.35/1.78	1.49	5.76	0.680	26.50	10.06

Table 3. Benchmarks with PCF=160

	Interferogram Length (points)			Benchmarks (s)				Total Time (s)	
Scan Mode	**Short Wing**	**Long Wing**	**Total**	**I/O(R/W)**	**Interpolation**	**Phase Correction**	**FFT**	**Acquisition**	**Processing**
SED 850 Band	180	180	360	0.19/0.11	0.17	0.88	0.045	1.80	1.40
SED Dual Band	300	300	600	0.21/0.15	0.28	1.26	0.057	3.00	1.96
Spectral Line 850 Band	180	3000	3180	0.27/1.04	0.85	6.48	0.406	15.90	9.05
Spectral Line Dual Band	300	5000	5300	0.36/1.75	1.39	10.54	0.670	26.50	14.71

6. External Java Libraries

The FTS-2 engine uses four external Java libraries: a Java version of FFTPack, Jama, StarJava, and a Java version of DRAMA.

The Java version of FFTPack has been translated from the original FORTRAN code by the FTS-2 group. FFTPack is a well-known package of FORTRAN subprograms for the fast Fourier transform of periodic and other symmetric sequences. It includes complex, real, sine, cosine, and quarter-wave transforms. The new Java version of FFTPack can now be downloaded from many public domains, such as:
http://netlib.bell-labs.com/netlib/fftpack/jfftpack.tgz
http://www.netlib.org/fftpack/jfftpack.tgz

JAMA is a basic linear algebra package for Java. It is developed by NIST and can be downloaded from:
http://math.nist.gov/javanumerics/jama/

Two other required Java libraries are StarJava (the latest version in Starlink CVS repository) and DJAVA (Java version of DRAMA). The FTS-2 Java DR Engine uses StarJava for I/O and the SOAP message interface since Apache Axis is integrated into StarJava. If DJAVA is used for the DRAMA message interface, the full DRAMA package must be installed.

If the user needs to run the Java program, TestSOAPClient.java, Globus' Java WS Core is also need. This Java package can be downloaded from
http://www-unix.globus.org/toolkit/survey/index.php?download=ws-core-4.0.2-bin.tar.gz

Appendix

A. Numerical Method of Phase Correction

A.1 Butterflied Phase Correction

The ideal interferogram can be expressed by:

$$I(x) = \int_0^\infty d\omega\ f(\omega)\cos(\omega x)$$

where, ω is the angular wave number ($\omega = 2\pi\sigma$) and x is the interferogram optical path difference (OPD). If the phase distortion is included, a real interferogram is:

$$I(x) = \int_0^\infty d\omega\, f(\omega)\cos(\omega x + \phi(\omega))$$

where, $\phi(\omega)$ is the phase distortion.

If the even symmetry of $f(\omega)$, $f(\omega) = f(-\omega)$, is taken into account, the interferogram can be rewritten as follows:

$$\begin{aligned} I(x) &= \int_0^\infty d\omega\, f(\omega)\cos(\omega x + \phi(\omega)) \\ &= \frac{1}{2}\int_0^\infty d\omega\, f(\omega)[e^{i(\omega x + \phi(\omega))} + e^{-i(\omega x + \phi(\omega))}] \\ &= \frac{1}{2}\int_{-\infty}^\infty d\omega [f(\omega)e^{i\varphi(\omega)}]e^{i\omega x} \end{aligned}$$

where, $\varphi(\omega) = \begin{cases} \phi(\omega), & \omega \ge 0 \\ -\phi(-\omega), & \omega < 0 \end{cases}$

$\phi(\omega)$ can be obtained by fitting the phase of double-sided part of the interferogram. $\varphi(\omega)$ is also called the butterflied phase.

The phase-corrected interferogram is defined as:

$$I'(x) = \int_0^\infty d\omega\ f(\omega)\cos(\omega x)$$

It is easy to show that:

$$I'(x) = I(x) \otimes FT^{-1}(e^{-i\varphi(\omega)})$$

where, $\otimes$ is the convolution operation and FT^{-1} is the inverse Fourier transform. The phase correction function is defined as:

$$PCF(x) = FT^{-1}(e^{-i\varphi(\omega)})$$

Owing to the symmetric properties of $\varphi(\omega)$:

$$\begin{aligned} PCF(x) &= C\int_0^\infty d\omega \cos(\omega x - \phi(\omega)) \\ &= C\left[\int_0^\infty d\omega \cos(\phi(\omega))\cos(\omega x) + \int_0^\infty d\omega \sin(\phi(\omega))\sin(\omega x)\right] \end{aligned}$$

where, C is the normalization constant of Fourier transform.

A.2 Numerical Method

In our numerical method, we can use the following discrete Fourier transform (DFT). The forward DFT of a 1-d complex array X of size *n* computes an array Y, where:

$$Y_k = \sum_{j=0}^{n-1} X_j e^{-2\pi jk\sqrt{-1}/n} .$$

The backward DFT computes:

$$Y_k = \sum_{j=0}^{n-1} X_j e^{2\pi jk\sqrt{-1}/n} .$$

For those who like to think in terms of positive and negative frequencies, this means that the positive frequencies are stored in the first half of the output and the negative frequencies are stored in backwards order in the second half of the output, i.e., the frequency -*k*/*n* is the same as the frequency (*n*-*k*)/*n*.

Without loss of generality, we assume that the maximum OPD of the interferogram is L and the size is N. In a discrete scheme with size *N*, $f(x_i)$, $x_i = i\dfrac{L}{N}$, $i = 0,\cdots,N-1$; $F(\omega_j) = FT(f(x))$, $\omega_j = j\dfrac{2\pi}{L}$, $j = 0,\cdots,N-1$.

B. Self-Adaptive Phase Correction

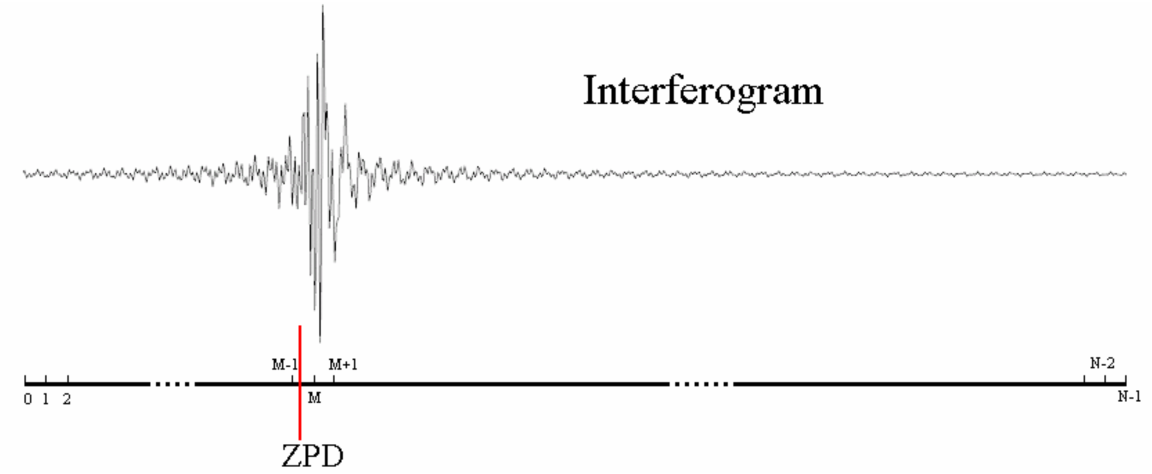

Fig.B.1. A discrete interferogram with N evenly-sampled points. The top curve represents an interferogram and the below bar represents the positions of the sampling points. The red vertical bar represents the exact position of zero-path distance (ZPD).

Without loss of generality, a position is represented by an index number, i.e., the physical position represented by L is (L x Δx + the physical position of position 0), where Δx is the interval between the two adjacent points. In Fig.1, ZPD falls in [M-1, M]. That is, there are M points in the left of ZPD and (N-M) points to the right of ZPD.

In the FTS-2 data reduction Java engine, DRPipeline, there are three input parameters, dsSize, ssSize, pcfSize_h to define the phase correction of an interferogram. When M ≥ dsSize, the points from (M - dsSize) to (M + dsSize - 1) are used as a short double-side interferogram to calculate the phase error. When M ≥ dsSize but dsSize does not satisfy the form, 2^a x 3^b x 5^c x 7^d (where a, b, and c are an arbitrary non-negative integer, d is 0 or 1), or M < dsSize, dsSize will be replaced with a new value, new_dsSize, which is the biggest number that is less than M and has the form, 2^a x 3^b x 5^c x 7^d, where a, b, and c are an arbitrary non-negative integer, d is 0 or 1.

When (N – M - pcfSize_h + 1) ≥ ssSize, the value of ssSize will keep unchanged. The phase correction will output a phase-corrected single-side interferogram with length = (ssSize + 1). When (N – M - pcfSize_h + 1) ≥ ssSize but ssSize does not satisfy the form, 2^a x 3^b x 5^c (where a, b, and c are an arbitrary non-negative integer), or (N – M - pcfSize_h + 1) < ssSize, ssSize will be replaced with a new value, new_ssSize, which is the biggest number that is less than (N-M-pcfSize_h + 1) and has the form, 2^a x 3^b x 5^c, where a, b, and c are an arbitrary non-negative integer. Then, the phase correction will output a phase-corrected single-side interferogram with length = (new_ssSize + 1).