DATA CAPTURE SYSTEM (DCS)

Software Design Description DCS-104

Release 1.4

Version 5.0

December 2002
## Document Change Summary

### List of Effective Pages

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Original</td>
</tr>
<tr>
<td>Sections 4.1.1.7.13</td>
<td>CTS to DDS Raw Data Transfer Via Gui</td>
</tr>
<tr>
<td>Sections 4.1.1.7.1</td>
<td>Change to DCS_GUI – Make selected file stand out</td>
</tr>
<tr>
<td>Sections 4.1.1.7.1</td>
<td>DCS_GUI (icons greyed out)</td>
</tr>
<tr>
<td>Sections 4.1.1.7.1, 4.1.1.7.10, 4.1.1.7.12, 4.3.2.3.5</td>
<td>DCS – Add capability to flag IGS validation and exchange data</td>
</tr>
<tr>
<td>Section 5.3</td>
<td>DCS – Add a Moving Window Display during data capture</td>
</tr>
<tr>
<td>Section 5.1.5.2</td>
<td>Allow OPS to Configure DCS READY Status for Any Ground Station</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change Number</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1.7.12</td>
<td>1943 – DCS - Incorrect Archive Status</td>
</tr>
<tr>
<td>3.3.2, 3.1.1.2.5, 3.1.1.7.2, 4.1.2.5, A.15</td>
<td>2000 – DCS -- add capability for a CTS to back-up another CTS.</td>
</tr>
<tr>
<td>3.1.1.5.2, 3.1.2.2, A.10</td>
<td>2006 – DCS -- FILE_SIZE field in DB does not work for ASN data.</td>
</tr>
<tr>
<td>3.1.1.2.5, 3.1.1.7.2, A.6, A.9</td>
<td>2063 – CTS3 Tape Drive (tps1d4) always shows a READY status even with no tape in it.</td>
</tr>
<tr>
<td>3.1.1.7.15</td>
<td>2067 – DCS Initialize Tape GUI Refresh Problem</td>
</tr>
<tr>
<td>3.1.1.5.2</td>
<td>2076 – DCS - Disable Recapture of RAW file if previous capture is not HOLD or SUCCESS</td>
</tr>
<tr>
<td>3.1.1.2.1</td>
<td>2135 – DCS - core dump -core is an invalid schedule file,</td>
</tr>
<tr>
<td>2.5</td>
<td>2143 – DCS - PR Moving Window Display</td>
</tr>
<tr>
<td>3.1.2.1</td>
<td>2170 – DCS - LCTS2 Capture failure with unable to open device error</td>
</tr>
</tbody>
</table>

### Document History

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Status/Issue</th>
<th>Publication Date</th>
<th>Change Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9/10/09 - iii - DCS 104.4
<table>
<thead>
<tr>
<th>4.0</th>
<th>Original</th>
<th>June 2002</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>DCS 104.3</td>
<td>June 2002</td>
<td>CCR 1618</td>
</tr>
<tr>
<td>4.0</td>
<td>DCS 104.3</td>
<td>June 2002</td>
<td>CCR 1662</td>
</tr>
<tr>
<td>4.0</td>
<td>DCS 104.3</td>
<td>June 2002</td>
<td>CCR 1677</td>
</tr>
<tr>
<td>4.0</td>
<td>DCS 104.3</td>
<td>October 2002</td>
<td>CCR 1904</td>
</tr>
<tr>
<td>4.0</td>
<td>DCS 104.3</td>
<td>October 2002</td>
<td>CCR 1996</td>
</tr>
<tr>
<td>5.0</td>
<td>DCS 104.4</td>
<td>December 2002</td>
<td>CCR 2068</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5.0</th>
<th>DCS 104.4</th>
<th>December 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCR 2000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR 2006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR 2063</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR 2067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR 2076</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR 2135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR 2143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCR 2170</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

Document Change Summary ........................................................................................................ iii
Table of Contents ......................................................................................................................... v
List of Figures ............................................................................................................................ vii
List of Tables .............................................................................................................................. ix

## Section 1 INTRODUCTION ........................................................................................................ 10
1.1 Identification ......................................................................................................................... 10
1.2 System Overview .................................................................................................................. 10
1.3 Document Overview ............................................................................................................. 10

## Section 2 Design Considerations .............................................................................................. 11
2.1 Assumptions and Dependencies .......................................................................................... 11
2.2 General Constraints .............................................................................................................. 11
2.3 Conventions .......................................................................................................................... 11
2.4 Methodology ......................................................................................................................... 11
2.5 System Architecture ............................................................................................................ 11
2.6 Other Design Considerations ............................................................................................... 13

## Section 3 Architectural Design ................................................................................................ 14
3.1 CSCI components ................................................................................................................ 14
3.1.1 DCS Database Server (DDS) ................................................................................... 14
3.1.2 Capture Transfer Subsystem (CTS) .......................................................................... 61
3.1.3 Capture Transfer Display (CTD) ............................................................................. 68
3.2 Concept of execution ............................................................................................................ 68
3.2.1 Normal Raw Data Flow ............................................................................................. 68
3.3 Interface design .................................................................................................................... 69
3.3.1 Interface identification and diagrams ........................................................................ 69
3.3.2 DCS to CTS Interface ............................................................................................... 69
3.3.3 DCS to Destination Systems Interface ..................................................................... 76
3.3.4 MOC to DDS Interface ............................................................................................. 76
3.3.5 RDC to MWD Formatter Interface .......................................................................... 76
3.3.6 CTS to CTD Interface ............................................................................................... 79

## Section 4 CSCI Detailed Design ................................................................................................ 81
4.1 DCS Database Server ........................................................................................................... 81
4.1.1 Database Component .................................................................................................... 81
4.1.2 Monitor and Control Component ................................................................................. 81
4.1.3 Back-up Archive Component ....................................................................................... 99
4.1.4 Delete Raw Files Component (DDS) .......................................................................... 101
4.1.5 Restage from Tape Component ................................................................................... 103
4.1.6 Journaling Component ................................................................................................. 110
4.2 Capture Transfer Subsystem ............................................................................................. 111
4.2.1 AutoCapture Component ............................................................................................ 111
4.2.2 Raw Data Capture Component .................................................................................... 118
4.2.3 Transfer To Tape Component ....................................................................................... 120
4.2.4 Delete Raw Files Component (CTS) ......................................................... 125
4.2.5 Raw File Transfer Component (CTS) .................................................... 127
4.2.6 Raw Data Transmit Component ............................................................ 130
4.2.7 Journaling Component ......................................................................... 131
4.3 MWD Formatter Component .................................................................... 131
4.3.1 Purpose .................................................................................................. 131
4.3.2 Subcomponent Design .......................................................................... 133
4.4 Moving Window Display Component ........................................................ 190

Section 5 Requirements Traceability .............................................................. 191

Section 6 Notes .......................................................................................... 199
6.1 Raw Data File Queue ............................................................................... 199

Appendix A APPENDIXES ........................................................................... 201

A DDS_OPS_DB Data Dictionary ................................................................... 201
A.1 MISSION_ACCT ........................................................................................ 201
A.2 MISSION_STATION_ACCT ................................................................. 201
A.3 DESTINATION_ACCT ............................................................................ 201
A.4 ROUTING_ACCT .................................................................................... 202
A.5 CONTACT_SCHEDULES ........................................................................ 202
A.6 DCS_CAPTURE_ACCT ............................................................................ 203
A.7 CAPTURE_MISSION_ACCT ................................................................. 204
A.8 DCS_CONTACT_SCHED_FILES ......................................................... 204
A.9 DCS_CONFIGURATION ............................................................................ 205
A.10 DCS_RAWFILE_SCHED_FILES ......................................................... 206
A.11 TRANSFER_ACCT ............................................................................... 207
A.12 TRANSFER_ACCT_ARCHIVE ............................................................. 208
A.13 BACKUP_ACCT ................................................................................... 208
A.14 TEMP.CTS_FILES ............................................................................... 209
A.15 DCS_CAPT_CHAN_MAP ...................................................................... 209

Appendix B Acronyms .................................................................................. 210

Referenced Documents ................................................................................. 211
List of Figures

Figure 2-1 DCS Environment ......................................................... 12
Figure 3-1 DCS Database Subsystem ............................................. 15
Figure 3-2 DCS Database Schema ................................................ 16
Figure 3-3 mac_DDS ................................................................. 23
Figure 3-4 mac_IngestContact and mac_GenSuptSchedule ............... 25
Figure 3-5 mac_SendSetup ......................................................... 27
Figure 3-6 mac_ManCapture ..................................................... 28
Figure 3-7 mac_GetCTSFFileList ............................................... 29
Figure 3-8 mac_BackupArchive and mac_DeleteRawFiles ................. 30
Figure 3-9 mac_Restage and mac_UpdDCSAcct ............................. 32
Figure 3-10 mac_DDS_gui Setup Menu Activations ......................... 35
Figure 3-11 mac_DDS_gui Control Menu Activations .................... 36
Figure 3-12 mac_DDS_gui Button Activations .............................. 37
Figure 3-13 DCS Main Window .................................................. 38
Figure 3-14 Set Capture Parameters ........................................... 41
Figure 3-15 Capture Channel Map ............................................. 43
Figure 3-16 Edit Contact Schedules .......................................... 44
Figure 3-17 Generate Tape Label ............................................. 45
Figure 3-18 Ingest Contact Schedules ...................................... 46
Figure 3-19 Manual Data Capture ........................................... 48
Figure 3-20 Start Copy to Tape ............................................. 50
Figure 3-21 Start Restage ....................................................... 51
Figure 3-22 DDS Configuration GUI ...................................... 53
Figure 3-23 Mission Configuration GUI .................................. 54
Figure 3-24 Destination System Configuration GUI .................... 55
Figure 3-25 Raw File Details GUI ........................................ 57
Figure 3-26 Transfer History GUI ........................................... 58
Figure 3-27 Transfer File GUI .............................................. 59
Figure 3-28 Initialize Tape .................................................... 60
Figure 3-29 Capture Transfer Subsystem .................................. 62
Figure 4-1 mac_DDS Flowchart ............................................ 84
Figure 4-2 Check_for_schedule Flowchart .................................. 85
Figure 4-3 Check_disk_space flowchart .................................... 86
Figure 4-4 Check_for_xfers Flowchart .................................... 87
Figure 4-5 mac_IngestContact Flowchart ................................ 90
Figure 4-6 mac_GenSuptSchedule Flowchart ............................... 93
Figure 4-7 mac_BackupArchive Flowchart ................................ 101
Figure 4-8 mac_DeleteRawFiles Flowchart ................................ 103
Figure 4-9 mac_Restage Flowchart ....................................... 106
Figure 4-10 mac_UpdDCSAcct Flowchart ................................ 109
Figure 4-11 rdc_AutoCapture Flowchart .................................. 114
Figure 4-12 rdc_AutoTransfer Flowchart .................................. 117
Figure 4-13 rdc_Capture Flowchart ..................................... 120
Figure 4-14 rdc_Save Flowchart ................................................................. 123
Figure 4-15 rdc_DeleteFiles Flow Chart .................................................. 127
Figure 4-16 rdc_TransferFile Flow Chart .................................................. 129
Figure 4-17 Processing String ................................................................. 133
Figure 6-1 Raw Data Queue ................................................................. 200
## List of Tables

Table 1: MISSION_ACCT ........................................................................................................ 201  
Table 2: MISSION_STATION_ACCT .................................................................................. 201  
Table 3: DESTINATION_ACCT .......................................................................................... 201  
Table 4: ROUTING_ACCT .................................................................................................. 202  
Table 5: CONTACT_SCHEDULES ...................................................................................... 202  
Table 6: DCS_CAPTURE_ACCT .......................................................................................... 203  
Table 7: CAPTURE_MISSION_ACCT .................................................................................... 204  
Table 8: DCS_CONTACT_SCHED_FILES .............................................................................. 204  
Table 9: DCS_CONFIGURATION .......................................................................................... 205  
Table 10: DCS_RAWFILE_ACCT ......................................................................................... 206  
Table 11: TRANSFER_ACCT .............................................................................................. 207  
Table 12: TRANSFER_ACCT_ARCHIVE .............................................................................. 208  
Table 13: BACKUP_ACCT .................................................................................................. 208  
Table 14: TEMP_CTS_FILES ............................................................................................. 208  
Table 15: DCS_CAPT_CHAN_MAP ....................................................................................... 209
Section 1  INTRODUCTION

1.1 Identification

The purpose of this document is to present the detailed software design for the Data Capture System (DCS). The DCS was developed for the United States Geological Survey’s (USGS) Earth Resources Observation System (EROS) Data Center (EDC) of the United States Department of Interior (DOI) for operation at the EDC.

1.2 System Overview

The DCS was developed to be a modular wideband data capture platform. As a modular platform, the DCS will be used to capture data on multiple missions. Therefore, whenever possible, the DCS design is defined without regard to a specific mission.

1.3 Document Overview

This SDD describes the DCS software structure, the software components, the interfaces and data necessary for DCS implementation. Every requirement in the DCS SRS is traceable to one or more design entities in this SDD. This document comprises seven sections and one appendix:

- **Section 1** contains the introduction, the system overview, and the document overview.
- **Section 2** presents design considerations/decisions.
- **Section 3** describes the CSCI architectural design.
- **Section 4** provides a detailed description of each software component.
- **Section 5** contains traceability from each software unit to the CSCI requirements allocated to it. Also presented is traceability from each CSCI requirement to the software units to which it is allocated.
- **Section 6** contains general information that aids in understanding this document.
Section 2 Design Considerations

2.1 Assumptions and Dependencies

The system design set forth in this document assumes the following:
- Changes to existing modules and creation of new modules will be to support required new functionality.

2.2 General Constraints

In addition to the software and hardware constraints outlined in the DCS Software Requirements Specification (SRS) (Applicable Document 0) the development of this software is constrained by the following:

2.3 Conventions

The file and module naming convention will follow the convention used in the DCS software that is being reused, namely:
- All file and module names on the CTS will follow the same convention used in RDCS (i.e. names will begin with 'rdc_'). The CTS will not need database access routines; therefore the rdc_db_XXX routines will be renamed to mac_db_XXX routines if they are reused on the DDS. Processes built for the CTS that are reused on the DDS will be reused without a name change.
- A portion of the global library routines originally reused from LPS are reused on the DDS and CTS. All file and module names reused without any modification will follow the same convention used on LPS (i.e. names will begin with 'lps_' or 'lps_db'). New or modified global routines will have a naming convention where the file and module names begin with 'dcs_'.

2.4 Methodology

The approach for the DCS software design is to leverage off existing DCS software. The code itself serves as a design to be modified.

2.5 System Architecture

Refer to Error! Reference source not found. for the DCS Environment in Figure 2-1. The DCS has two basic subsystems: the DCS Database Subsystem (DDS) and one or more Capture Transfer Subsystems (CTS). The DDS serves as the primary operator station for the entire DCS and also tracks raw file delivery to destination systems and/or to back-up tapes (for contingencies). Each CTS also can have an optional Capture Transfer Display (CTD) subsystem which can be used to show a Moving Window Display of imagery during raw data capture.
Each CTS will receive and temporarily store one to four raw wideband data streams obtained from a receiving station (such as a satellite receiving station). The raw data streams will be captured onto the subsystem’s local storage using one or two high-speed dual channel serial capture devices. During capture, the raw data is also buffered and made available for an optional Moving Window Display (MWD).

The Moving Window Display is useful for monitoring data quality during capture. The CTD displays video imagery as the CTS captures it to disk. The CTD can also re-distribute the video to other, remotely connected MWDs.

After the wideband data is captured, the CTS will transfer the data to the DDS for distribution to mission-defined destination systems. Destination systems are defined as either processing or archiving systems. The main difference to DCS is that archiving systems will permanently store the raw data, whereas the processing systems do not. This is important for DCS automated and manual clean-up operations. DCS will ensure that either all archiving systems have received a raw file or the file is backed up to tape before it is automatically deleted from local storage. Processing systems that need to process a file that is no longer in the DCS inventory (on-line or on tape) will need to request the tape from the corresponding mission’s archiving system.
Each CTS will also have the ability to function stand-alone (i.e. without the DDS). In stand-alone mode, operational setup is performed via editing text files and the wideband data will be transferred to tape for transport. These tapes can then be ingested onto the DDS for distribution to the destination system(s).

The DDS will be used to temporarily store the captured wideband data for retrieval by the destination systems. The wideband data will be received from the CTS(s) via network transfer or from tape (for ingest of data from the CTSs operating in stand-alone mode). Normally, the raw data is then transferred by the appropriate network-connected destination system(s). The transfer is initiated and carried out by the destination systems. DDS operators can also copy selected raw data to tape in atypical operation.

Each mission will have a mission operations center (MOC) that will deliver mission-specific contact schedules to the DDS. The contacts defined in the schedule are for live automated captures at EDC. No data is transferred from the DDS to the MOCs.

2.6 Other Design Considerations

Since the DCS design was leveraged off existing LPS code, there are quite a few constants/functions, etc. that are named for the LPS architecture and thus have “LPS_” in the name. Those constants and functions that are re-used without modification will not be renamed.

The DCS design also incorporates re-used MWD Formatter code written in C++ and MWD server code written in Delphi.
Section 3  Architectural Design

3.1  CSCI components

The Data Capture System software is divided into four computer software configuration items (CSCIs): the Monitor and Control Subsystem (MACS), the Raw Data Capture Subsystem (RDCS), the Moving Window Display Formatter, and the Moving Window Display server. The MACS will execute on the DDS, the RDCS and MWDFormatter will execute on the CTS, and the MWD will execute on the CTD.

3.1.1  DCS Database Server (DDS)

The DCS Database Server (DDS) is responsible for making raw data available to processing systems and archive systems. The DDS is also the main operator station for controlling raw data capture operations.
Figure 3-1 shows the top-level data flow for the DDS. The following paragraphs explain each of the processes in more detail.

**Figure 3-1 DCS Database Subsystem**

### 3.1.1.1 Database Component

The database component is defined below. For complete details of the DCS database, refer to applicable document #0.
The DCS Database will be DDS_OPS_DB database, a stand-alone Oracle schema on the DDS. The DCS Database schema is shown in Figure 3-2. Appendix A contains the data dictionary for DDS_OPS_DB.

3.1.1.1.1 Table MISSION_ACCT

3.1.1.1.1.1 Purpose

Mission Identification is defined in table MISSION_ACCT. The table lists each mission supported by DCS, along with the directory path for raw files and the default priority assignment.

Only privileged operators will be able to modify this table.

Requirements defined in SRS 3.1.2.1 are allocated to this table.

3.1.1.1.2 Development Status

Table MISSION_ACCT will be reused without change.
3.1.1.1.3 Resource Utilization
The resource utilization for table MISSION_ACCT is expected to remain trivially small (only 1 record per supported mission/data_type combination).

3.1.1.1.4 Program Library
Table MISSION_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.2 Table DESTINATION_ACCT

3.1.1.2.1 Purpose
Destination System Identification is defined in table DESTINATION_ACCT. The table lists each raw data file destination system that will retrieve data from DCS. Only privileged operators will be able to modify this table. Requirements defined in SRS 3.1.2.2 are allocated to this table.

3.1.1.2.2 Development Status
Table DESTINATION_ACCT will be reused without change.

3.1.1.2.3 Resource Utilization
The resource utilization for table DESTINATION_ACCT is expected to remain trivially small (only 1 record per supported destination system).

3.1.1.2.4 Program Library
Table DESTINATION_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.3 Table ROUTING_ACCT

3.1.1.3.1 Purpose
Raw data file routing information is defined in table ROUTING_ACCT. The table lists all raw data file destination systems that will retrieve data from DCS by mission. All operators will be able to modify this table. Requirements defined in SRS 3.1.2.3 are allocated to this table.

3.1.1.3.2 Development Status
Table ROUTING_ACCT will be reused without change.
3.1.1.3.3 **Resource Utilization**

The resource utilization for table ROUTING_ACCT is expected to remain trivially small (only 1 record per supported mission/destination combination).

3.1.1.3.4 **Program Library**

Table ROUTING_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.4 **Table MISSION_STATION_ACCT**

3.1.1.4.1 **Purpose**

Ground stations assigned to each mission will be listed in table MISSION_STATION_ACCT. The table lists all ground station identifiers for each mission supported by DCS.

All operators will be able to modify this table.

Requirements defined in SRS 3.1.2.1 are allocated to this table.

3.1.1.4.2 **Development Status**

Table MISSION_STATION_ACCT will be reused without change.

3.1.1.4.3 **Resource Utilization**

The resource utilization for table MISSION_STATION_ACCT is expected to remain small (only a few stations for each supported mission).

3.1.1.4.4 **Program Library**

Table MISSION_STATION_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.5 **Table CONTACT_SCHEDULES**

3.1.1.5.1 **Purpose**

Contact schedule files will be ingested into table CONTACT_SCHEDULES. The table lists all scheduled downlinks for live supports at EDC.

All operators will be able to modify this table.

Requirements defined in SRS 3.1.3.1 are allocated to this table.

3.1.1.5.2 **Development Status**

Table CONTACT_SCHEDULES will be reused without change.
3.1.1.5.3 Resource Utilization

The resource utilization for table CONTACT_SCHEDULES is expected to remain small (probably only 48 hours worth of live supports for each supported mission).

3.1.1.5.4 Program Library

Table CONTACT_SCHEDULES is implemented in the DDS_OPS_DB schema.

3.1.1.6 Table DCS_CAPTURE_ACCT

3.1.1.6.1 Purpose

Capture Transfer System setup information is stored in table DCS_CAPTURE_ACCT. The table lists all capture systems controlled by the DDS.

All operators will be able to modify this table.

Requirements defined in SRS 3.1.1.2 are allocated to this table.

3.1.1.6.2 Development Status

Table DCS_CAPTURE_ACCT will be reused without change.

3.1.1.6.3 Resource Utilization

The resource utilization for table DCS_CAPTURE_ACCT is expected to remain trivially small (only 1 record per CTS).

3.1.1.6.4 Program Library

Table DCS_CAPTURE_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.7 Table CAPT_MISSION_ACCT

3.1.1.7.1 Purpose

The mission(s) supported by each Capture Transfer System are defined in table CAPT_MISSION_ACCT.

Only privileged operators will be able to modify this table.

Requirements defined in SRS 3.1.1.2.1.1 are allocated to this table.

3.1.1.7.2 Development Status

Table CAPT_MISSION_ACCT will be reused without change.
3.1.1.7.3 Resource Utilization

The resource utilization for table CAPT_MISSION_ACCT is expected to remain trivially small (only 1 record per supported mission/capture system combination).

3.1.1.7.4 Program Library

Table CAPT_MISSION_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.8 Table DCS_RAWFILE_ACCT

3.1.1.8.1 Purpose

The raw data files handled by DCS are listed in table DCS_RAWFILE_ACCT.
All operators will be able to modify this table.
Requirements defined in SRS 3.1.1.5 are allocated to this table.

3.1.1.8.2 Development Status

Table DCS_RAWFILE_ACCT will be reused without change.

3.1.1.8.3 Resource Utilization

The resource utilization for table DCS_RAWFILE_ACCT will be one record for each raw file for the duration of the project. For Landsat 7, there should be an average of 5.5 contacts per day and 4 files per contact (so, 22 files per day for at least 3 years, or approximately 24,000 records).

3.1.1.8.4 Program Library

Table DCS_RAWFILE_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.9 Table TRANSFER_ACCT

3.1.1.9.1 Purpose

The transfer of raw data files is tracked in table TRANSFER_ACCT.
All operators will be able to modify this table. In addition, the destination systems will be able to update the XFER_STATUS and STATUS_DATE fields.
Requirements defined in SRS 3.1.2.4 and 3.1.2.5 are allocated to this table.

3.1.1.9.2 Development Status

Table TRANSFER_ACCT will be reused without change.
3.1.1.9.3 Resource Utilization

The resource utilization for table TRANSFER_ACCT will be one record for each destination system that will retrieve a given raw file. For Landsat 7, there will be two destinations for the 22 files per day for at least 3 years.

3.1.1.9.4 Program Library

Table TRANSFER_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.10 Table BACKUP_ACCT

3.1.1.10.1 Purpose

Backup archival of raw data files is tracked in table BACKUP_ACCT.

All operators will be able to modify this table.

Requirements defined in SRS 3.2.1.2 are allocated to this table.

3.1.1.10.2 Development Status

Table BACKUP_ACCT will be reused without change.

3.1.1.10.3 Resource Utilization

Since the Backup/Archive is for non-nominal operations, the resource utilization for table BACKUP_ACCT is expected to be reasonably small (only 1 record per file on a DDS Backup/Archive tape).

3.1.1.10.4 Program Library

Table BACKUP_ACCT is implemented in the DDS_OPS_DB schema.

3.1.1.11 Table DCS_CONTACT_SCHED_FILES

3.1.1.11.1 Purpose

Contact Schedule filenames are temporarily stored in table DCS_CONTACT_SCHED_FILES.

All operators will be able to modify this table.

Requirements defined in SRS 3.2.1.4 are allocated to this table.

3.1.1.11.2 Development Status

Table DCS_CONTACT_SCHED_FILES will be reused without change.
3.1.1.11.3 Resource Utilization

The resource utilization for table DCS_CONTACT_SCHED_FILES is minimal. The table is used to temporarily hold contact schedule file names for manual schedule ingests and the ingest routine will move ingested files to a “processed” directory, so it is expected to be trivially small (only 1 record per file in the schedule directory).

3.1.1.11.4 Program Library

Table DCS_CONTACT_SCHED_FILES is implemented in the DDS_OPS_DB schema.

3.1.1.12 Table DCS_CONFIGURATION

3.1.1.12.1 Purpose

DCS configuration parameters are stored in table DCS_CONFIGURATION. All operators will be able to modify this table. Requirements defined in SRS 3.2.1.1 are allocated to this table.

3.1.1.12.2 Development Status

Table DCS_CONFIGURATION will be reused without change.

3.1.1.12.3 Resource Utilization

The resource utilization for table DCS_CONFIGURATION is trivial (only 1 record of DDS configuration options).

3.1.1.12.4 Program Library

Table DCS_CONFIGURATION is implemented in the DDS_OPS_DB schema.

3.1.1.13 Table TEMP_CTS_FILES

3.1.1.13.1 Purpose

CTS file information is temporarily stored in table TEMP_CTS_FILES. The table allows the GUI to access the list for operator selection for manual file transfer. All operators will be able to modify this table. Requirements defined in DCS SRS 3.2.1.6.12 are allocated to this table.

3.1.1.13.2 Development Status

Table TEMP_CTS_FILES will be new development.
3.1.1.13.3 Resource Utilization

The resource utilization for table TEMP_CTS_FILES is trivial (only 1 record of file information for each file currently on-line on a given CTS).

3.1.1.13.4 Program Library

Table TEMP_CTS_FILES will be implemented in the DDS_OPS_DB schema.

3.1.1.2 Monitor and Control Component

3.1.1.2.1 mac_DDS

![Diagram of mac_DDS](image)

**Figure 3-3 mac_DDS**

3.1.1.2.1.1 Purpose

The mac_DDS process runs on the DDS. The purpose of the mac_DDS process is to automatically start the mac_DDS_gui, the mac_IngestContact, mac_BackupArchive, and the mac_Delete processes at appropriate times (and depending on configuration options).

Figure 3-3 shows the software units comprising mac_DDS. mac_DDS will have three main functions:
• **Check_for_schedule** – if enabled, the Check_for_schedule function will periodically check for new contact schedule files. When a new file is detected, the routine will start an automatic ingest of the contact schedule (see 3.1.1.2). This function will be implemented in module mac_DDS.c

• **Check_disk_space** – if enabled, the Check_disk_space function will periodically check the local disk available space. When the available space falls below an operator tunable threshold, the module will automatically start backup (see 3.1.1.2.1.3) and file deletion (see 3.1.1.3.1.3) routines to prevent the local disk from overfilling.

• **Check_for_xfers** – if enabled, will check the database for completed raw file transfers. When a raw file has successfully been transferred to all destinations, the function will automatically start file deletion.

   All of the requirements delineated in paragraph 3.2.1.1 of the SRS are allocated to the mac_DDS process.

3.1.1.2.1.2 Development Status

CCR 2135 – mac_DDS polling processes will be updated to connect to the database before the looping and not connect/disconnect within the loop.

3.1.1.2.1.3 Resource Utilization

   The resource utilization for mac_DDS.c is not a concern.

3.1.1.2.1.4 Program Library

   Module mac_DDS.c will be implemented as a stand-alone process.

   Database access routines are in the MACS database library for retrieving and updating information in the database.
3.1.1.2.2 mac_IngestContact

3.1.1.2.2.1 Purpose

The mac_IngestContact process runs on the DDS Subsystem. The purpose of the mac_IngestContact process is to read contact schedule files and insert the information into the database.

See Figure 3-4 for a top-level context diagram of mac_IngestContact.

The requirements of paragraphs 3.2.1.1.1-3.2.1.1.4 of the SRS are allocated to the mac_IngestContact process.

After ingest, mac_IngestContact executes mac_GenSuptSchedule to deliver the updated CTS-specific support schedules to the affected CTS's.

3.1.1.2.2.2 Development Status

The mac_IngestContact process will be reused without change.

3.1.1.2.2.3 Resource Utilization

The resource utilization for mac_IngestContact is not a concern.
3.1.1.2.2.4 Program Library

Module mac_IngestContact is implemented as a stand-alone process.

3.1.1.2.3 mac_GenSuptSchedule

3.1.1.2.3.1 Purpose

The mac_GenSuptSchedule process runs on the DDS Subsystem. The purpose of the mac_GenSuptSchedule process is to create and send a support schedule to a Capture Transfer Subsystem.

See Figure 3-4 for a top-level context diagram of mac_GenSuptSchedule.

The requirements of paragraph 3.2.1.1.5 of the SRS are allocated to the mac_GenSuptSchedule process.

3.1.1.2.3.2 Development Status

The mac_GenSuptSchedule process will be reused without change.

3.1.1.2.3.3 Resource Utilization

The resource utilization for mac_GenSuptSchedule.c is not a concern.

3.1.1.2.3.4 Program Library

Module mac_GenSuptSchedule.c is implemented as a stand-alone process.

3.1.1.2.4 mac_InsertFileNames

3.1.1.2.4.1 Purpose

The mac_InsertFileNames process runs on the DDS Subsystem. The purpose of the mac_InsertFileNames process is to temporarily insert contact schedule filenames into the database for manual selection and ingest via the GUI.

The requirements of paragraph 3.2.1.1.4 of the SRS are allocated to the mac_InsertFileNames process.

3.1.1.2.4.2 Development Status

The mac_InsertFileNames process will be reused without change.

3.1.1.2.4.3 Resource Utilization

The resource utilization for mac_InsertFileNames.c is not a concern.

3.1.1.2.4.4 Program Library

Module mac_InsertFileNames.c is implemented as a stand-alone process.
3.1.1.2.5  mac_SendSetup

Figure 3-5  mac_SendSetup

3.1.1.2.5.1  Purpose
The mac_SendSetup process runs on the DDS. The purpose of the mac_SendSetup process is to create CTS setup information files and send them to the CTS.

Refer to Figure 3-5 for a top-level context diagram of mac_SendSetup.

The requirements of paragraph 3.2.1.1.7 of the SRS are allocated to the mac_SendSetup process.

3.1.1.2.5.2  Development Status
The mac_SendSetup process will be updated to create and send the Mission Information file and to retrieve the tape drive setting from the database.

3.1.1.2.5.3  Resource Utilization
The resource utilization for mac_SendSetup.c is not a concern.

3.1.1.2.5.4  Program Library
Module mac_SendSetup.c is implemented as a stand-alone process.
3.1.1.2.6 **mac_ManCapture**

**Purpose**

The mac_ManCapture process runs on the DDS Subsystem. The purpose of the mac_ManCapture process is to create a CTS Manual Capture Parameter file, send it to a CTS and signal the CTS to perform the manual capture.

See Figure 3-6 for a top-level context diagram of mac_ManCapture.

The requirements of paragraphs 3.2.1.1.6 and 3.2.1.1.8 of the SRS are allocated to the mac_ManCapture process.

**Development Status**

The mac_ManCapture process will be updated to include the following changes:

- CCR 1645 -- Dual Downlinks from Polar Tapes on the Same CTS
  - An optional “channel number” will be added to the command line parameters.
  - The “channel number”, if provided, will be added to the Manual Capture Parameter file. This channel number will be optional to override the CTS’s default calculation of a channel number.

**NOTE:** The “channel number” is used to uniquely identify different files in a dual downlink. Each CTS calculates a unique channel id based on the CTS’s host name. For live downlinks, each downlink is normally routed to a different CTS; however, for manual captures from wideband tape, it is sometimes desirable to capture both downlinks on one CTS.

**Resource Utilization**

The resource utilization for mac_ManCapture is not a concern.
3.1.1.2.7.7 mac_GetCTSFileList

3.1.1.2.7.7.1 Purpose

The mac_GetCTSFileList process runs on the DDS Subsystem. The purpose of the mac_GetCTSFileList process is to populate table TEMP_CTS_FILES with file names on a given CTS.

See Figure 3-6 for a top-level context diagram of mac_GetCTSFileList.

The requirements of paragraph 3.2.1.6.12 of the SRS are allocated to the mac_GetCTSFileList process.

3.1.1.2.7.2 Development Status

The mac_GetCTSFileList process will be new development.

3.1.1.2.7.3 Resource Utilization

The resource utilization for mac_GetCTSFileList is not a concern.

3.1.1.2.7.4 Program Library

Module mac_GetCTSFileList is implemented as a stand-alone process.
3.1.1.3 Back-up Archive Component

3.1.1.3.1 mac_BackupArchive

3.1.1.3.1.1 Purpose

The mac_BackupArchive process runs on the DDS Subsystem. The purpose of the mac_BackupArchive process is to copy on-line raw files to tape and delete the on-line file, along with the appropriate updates to the database records.

See Figure 3-8 for a top-level context diagram of mac_BackupArchive.

3.1.1.3.1.2 Development Status

The mac_BackupArchive will be reused without modification.

3.1.1.3.1.3 Resource Utilization

The resource utilization for mac_BackupArchive is not a concern.
3.1.1.4 Program Library

Module mac_BackupArchive is implemented as a stand-alone process.
Database access routines are needed in the MACS database library for performing
the needed updates.

3.1.1.4 Delete Raw Files Component (DDS)

3.1.1.4.1 mac_DeleteRawFiles

3.1.1.4.1.1 Purpose

The mac_DeleteRawFiles process will run on the DDS Subsystem. The purpose of
the mac_DeleteRawFiles process is to delete raw files from on-line storage,
along with the appropriate updates to the database records.
See Figure 3-8 for a top-level context diagram of mac_DeleteRawFiles.

3.1.1.4.1.2 Development Status

The mac_DeleteRawFiles will be reused without changes.

3.1.1.4.1.3 Resource Utilization

The resource utilization for mac_DeleteRawFiles is not a concern.

3.1.1.4.1.4 Program Library

Module mac_DeleteRawFiles is implemented as a stand-alone process.
Database access routines are needed in the MACS database library for performing
the needed updates.
3.1.1.5 Restage from Tape Component

3.1.1.5.1 mac_Restage

3.1.1.5.1.1 Purpose

The mac_Restage process runs on the DDS Subsystem. The purpose of the mac_Restage process is to copy raw files from DCS Backup/Archive tapes onto the local disk, along with the appropriate updates to the database records.

In order to ingest files archived on the CTS subsystems, which will be new data files to the DCS database, the process will have the ability to copy all files on a tape and register them. In order to restage a selected file (i.e. one file from a tape of registered files) the process will have the ability to copy just one selected file.

See Figure 3-9 for a top-level context diagram for mac_Restage.

All of the requirements delineated in paragraph 3.2.1.3 of the SRS is allocated to the mac_Restage process.
3.1.1.5.2 Development Status

CCR1678, mac_Restage will be updated to restage files to $DDS_RAWFILE_PATH/restage. Then when all of the restaged files are completely on-line, the files will be moved to the $DDS_RAWFILE_PATH directory to be registered into the DCS database.

3.1.1.5.3 Resource Utilization

The resource utilization for mac_Restage is not a concern.

3.1.1.5.4 Program Library

Module mac_Restage is implemented as a stand-alone process.

3.1.1.5.2 mac_UpdDCSAcct

3.1.1.5.2.1 Purpose

The mac_UpdDCSAcct process runs on the DDS Subsystem. The purpose of the mac_UpdDCSAcct process is to register raw files into the database records.

See Figure 3-9 for a top-level context diagram for mac_UpdDCSAcct.

Portions of the requirements delineated in paragraph 3.2.1.3 of the SRS are allocated to the mac_UpdDCSAcct process.

3.1.1.5.2.2 Development Status

CCR #1653, mac_UpdDCSAcct will be updated to check validity of the accounting file info (especially the file size) before ingest.

CCR #2076, mac_UpdDCSAcct will be updated to check for any XFER_STATUS = 'PROGRESS' before ingesting and moving a file.

CCR 2006 – mac_UpdDCSAcct will be updated to use an integer number of bytes for file size (.acct files will still use MB for backward compatibility).

3.1.1.5.2.3 Resource Utilization

The resource utilization for mac_UpdDCSAcct is not a concern.

3.1.1.5.2.4 Program Library

Module mac_UpdDCSAcct is implemented as a stand-alone process.
3.1.1.6 Journaling Component

3.1.1.6.1 mac_JournalFileEntry

3.1.1.6.1.1 Purpose

The mac_JournalFileEntry process runs on the DDS Subsystem. The purpose of the mac_JournalFileEntry process is to allow operators to insert custom journal messages into the DCS journals.

The process will insert the operator-supplied message into the DCS journal file.

All of the requirements delineated in paragraph 3.2.1.7 of the SRS are allocated to the mac_JournalFileEntry process.

3.1.1.6.1.2 Development Status

The mac_JournalFileEntry will be reused without changes.

3.1.1.6.1.3 Resource Utilization

The resource utilization for mac_JournalFileEntry is not a concern.

3.1.1.6.1.4 Program Library

Module mac_JournalFileEntry is implemented as a stand-alone process.

3.1.1.7 Graphical User Interface Component

The mac_DDS_gui windows will run on the DDS Subsystem. Figure 3-10, Figure 3-11 and Figure 3-12 show the software units comprising mac_DDS_gui. The green boxes represent individual forms (or windows) displayed on the DDS.

In Figure 3-10 the lines from the main window (mac_ui_DDS_main) to the other windows represent the main menu Setup actions that activate the child windows. Figure 3-11 shows the main menu Control actions.

In Figure 3-12 the lines from the main window (mac_ui_DDS_main) to the other windows represent the main window button actions that activate the child windows. The main window buttons perform actions for the selected raw data file.
Figure 3-10 mac_DDS_gui Setup Menu Activations
Figure 3-11 mac_DDS_gui Control Menu Activations
Figure 3-12 mac_DDS_gui Button Activations
3.1.1.7.1 mac_ui_DDS_main

Figure 3-13 DCS Main Window

Figure 3-13 shows the mac_ui_DDS_main window, which is titled “Data Capture System”.

3.1.1.7.1.1 Purpose

The purpose of the mac_ui_DDS_main window is to provide operations with a graphical user interface for setup and control of DCS operations. Summary status information of active raw files is presented in the main window with additional buttons for common actions. Additional setup and control is implemented via a drop-down menu at the top of the window.

The window displays:

- **File Name** – filename of captured file. If the file is active the background color will be based on the priority. The currently selected file has a black background with yellow text color scheme.

  If the file is inactive, the background color will be gray. Files will be listed in “queue” order (i.e. highest priority files from oldest to youngest, then next highest priority from oldest to youngest, etc. down to the inactive files). This
value is display-only and is retrieved from field RAW_DATA_FILE_NAME of table DCS_RAWFILE_ACCT.

Note: “inactive” means the file is no longer on-line and no longer in the DCS Backup Archive.

- **Vol (Mb)** – file volume (size). Color scheme will follow the **File Name** box. This value is display-only and is retrieved from field FILE_SIZE of table DCS_RAWFILE_ACCT.
- **Data Type** – raw data file type (e.g. NOM for Nominal Data, VAL for Validation Data, EXC for Exchange Data).
- **Priority** – handling priority of file. Color scheme will follow the **File Name** box. This is a drop-down list that will allow updating the PRIORITY field of table DCS_RAWFILE_ACCT.
- **Archived** – delivery status of the file to archiving system(s). **Yes** will indicate that all intended archiving systems have successfully retrieved the file. **No** will indicate that no archiving systems have retrieved it. **Par** will indicate that some archiving systems have retrieved it (occurs only when more than one processing system is defined in the database). If any archiving system has failed to retrieve the file, the background will be red, the color scheme will follow the **Raw Data File Name** field. This value is display-only and is summary information retrieved from field XFER_STATUS of table TRANSFER_ACCT (and ARCHIVE_FLAG of DESTINATION_ACCT).
- **Processed** – delivery status of the file to processing system(s). **Yes** will indicate all intended processing systems have successfully retrieved the file. **No** will indicate that no processing systems have retrieved it. **Par** will indicate that some processing systems have retrieved it (occurs only when more than one processing system is defined in the database). If any processing system has failed to retrieve the file, the background will be red, otherwise the color scheme will follow the **Raw Data File Name** field. This value is display-only and is summary information retrieved from field XFER_STATUS of table TRANSFER_ACCT (and ARCHIVE_FLAG of DESTINATION_ACCT).

The **Information** button will display the window shown in Figure 3-25, which gives full details about the selected file.

The **Delete** button will display a file deletion confirmation window for the selected file, and then execute file deletion (see section 3.1.1.4)

The **Backup** button will display a Backup/Archive file confirmation window for the selected file, and then execute backup/archive (see section 3.1.1.3).

The **Restage** button will display a Restage file confirmation window for the selected file (see Figure 3-21 Start Restage), and then execute restage for the file (see section 3.1.1.5).
The **Find File** button will refresh the main window with files that have substrings matching the search window string.

The **Show Inactive** checkbox, when checked, will refresh the main window to display all files, including inactive files. When unchecked, it will refresh the main window to only display active files.

All of the requirements delineated in paragraph 3.2.1.6 of the SRS are allocated to the `mac_ui_DDS_main`.

### 3.1.1.7.1.2 Development Status

The `mac_ui_DDS_main` window was updated for DCS 1.2 with the following CCRs:

- **CCR 1666** - Find File for DCS_GUI
  An edit box and Find button will be added to narrow the listing to files matching a pattern.

- **CCR 1678** – Restage clarification
  A phrase such as “File Actions” will be added above the file action buttons (Information, Delete, Backup, and Restage). Restage in the Control menu will be changed from “Restage” to “Ingest Tape”.

- **CCR 1662** -- Change to DCS_GUI -- make selected file stand out better.
  The selected file will have a white background.

- **CCR 1677** -- DCS_GUI (icons greyed out)
  A modification will be made to re-enable the buttons after they’ve been disabled (via selecting a different file or a blank line).

### 3.1.1.7.1.3 Resource Utilization

The resource utilization for `mac_ui_DDS_main` is not a concern.

### 3.1.1.7.1.4 Program Library

Module `mac_ui_DDS_main` is implemented as a stand-alone Oracle form.
3.1.1.7.2 mac_ui_edit_capt_parms

Figure 3-14 shows the mac_ui_edit_capt_parms window, which is titled “Set Capture Parameters”.

3.1.1.7.2.1 Purpose

The purpose of the mac_ui_edit_capt_parms window is to provide operations with a graphical user interface to set operational parameters for individual CTS operations (see Table 6: DCS_CAPTURE_ACCT in Appendix A).

The **Capture System** drop-down box will allow selection of the individual CTS systems defined in table DCS_CAPTURE_ACCT (field CAPT_SYS_ID).

The **Software Version** displays the software version number of the CTS software (see Table 6: DCS_CAPTURE_ACCT field SOFTWARE_VER_NUM).

The **IP Address** edit box displays the Internet Protocol address of the DDS (see Table 9: DCS_CONFIGURATION field DCS_HARDWARE_STRING_ID).

Comment [bjp1]: Page 1
Need to add: suspend/isolate, sched, setup, capture directories, delay/delay time option, ftp login info and capture sources.
The **User Name** edit box displays the ftp login name for the CTS file transfers (see Table 6: DCS_CAPTURE_ACCT field USER_NAME).

The **Password** edit box displays the ftp login password for the CTS file transfers (see Table 6: DCS_CAPTURE_ACCT field PASSWORD).

The **Schedule Dir.** edit box displays the directory to receive CTS support schedule files (see Table 6: DCS_CAPTURE_ACCT field SCHEDULE_DIR).

The **Parameter Dir.** edit box displays the directory to receive CTS parameter files (see Table 6: DCS_CAPTURE_ACCT field PARAMETER_DIR).

The **Tape Drive** drop-down list allows the operator to select the tape drive for performing CTS transfers to tape (NOTE: the default selection is always the first drive found!).

The **Transfer Option** drop-down list will allow selection of the TRANSFER_OPTION field (see Table 6: DCS_CAPTURE_ACCT field TRANSFER_OPTION for valid values).

The **Idle Time** specifies how much idle time before the next scheduled capture is needed in order to start the transfer (see Table 6: DCS_CAPTURE_ACCT field XFER_IDLE_TIME).

The **Delete the raw file after transfer** checkbox allows the operator to set the DELETE_RAW_FILE field of the DCS_CAPTURE_ACCT.

The **Moving Window Display** edit allows the operator to set the MWD_NAME field of the DCS_CAPTURE_ACCT.

After the options are set, clicking the OK button will cause the options to be updated in the database and also to be sent to the corresponding CTS (see Figure 3-5 mac_SendSetup).

Requirements delineated in paragraph 3.2.1.6.11 of the SRS are allocated to the mac_ui_edit_capt_parms.

3.1.1.7.2.2 Development Status

The mac_ui_edit_capt_parms was updated with the following changes:

- **CCR 2000 - DCS** -- add capability for a CTS to back-up another CTS.
  
The Channel Map button was added to bring up the Capture Channel Map window shown in Figure 3-15. This window allows assigning a permanent channel id (aka “capture source”) to each capture device slot found in the CTS. The Channel can be ‘0’-‘9’, or ‘*’ (which means automatic assignment based on the CTS number, the number of capture devices found in the system, and the number of CTS’s at the ground station). The **Reset Map** button re-assigns ‘*’ for the Channel.

Comment [bjp2]: When autocapture is separated from transfer, this parameter could be deleted.
• **CCR 2063** - CTS3 Tape Drive (tps1d4) always shows a READY status—even with no tape in it. The current CTS tape drive setting is retrieved by `mac_ListTapeDrives` and placed in the database. The `mac_ui_edit_capt_parms` GUI will retrieve the current setting from the database and display it in the **Tape Drive** drop-down list.

### 3.1.1.7.3 Resource Utilization

The resource utilization for `mac_ui_edit_capt_parms` is not a concern.

### 3.1.1.7.4 Program Library

Module `mac_ui_edit_capt_parms` will be implemented as a stand-alone Oracle form.

### 3.1.1.7.3 `mac_ui_edit_cont_sched`

Figure 3-16 Edit Contact Schedules shows the Edit Contact Schedules form. Figure 3-10 shows how this form is activated.
3.1.1.7.3.1 Purpose

The purpose of the mac_ui_edit_cont_sched window is to provide operations with a graphical user interface for editing mission contact schedules.

The Scheduled Start Time and Scheduled Stop Time fields display the expected beginning and ending times for automated wideband raw data capture (fields SCHEDULED_START_TIME and SCHEDULED_STOP_TIME of table CONTACT_SCHEDULES).

The Mission ID is a drop-down list to select the contact schedule for a mission. The default mission displayed is the alphabetically first mission of all missions of the highest priority.

The Priority field is a drop-down list that displays the assigned priority for the capture. Operations can modify this value before a scheduled pass for the captured files to receive a different priority.

NOTE: When a contact schedule file is ingested, the default priority defined in table MISSION_ACCT is given to all contacts. If an upcoming pass defined in the database was modified to have a different priority, a warning message will be logged during ingest indicating that the pass will need to be updated by the operator. Also, when the contact information is
sent to the capture system, only the earliest AOS, the latest LOS times, and the highest priority is used for all capture systems.

Requirements delineated in paragraph 3.2.1.6.5 of the SRS are allocated to the mac_ui_edit_cont_sched.

3.1.1.7.3.2 Development Status
The mac_ui_edit_cont_sched is reused without change.

3.1.1.7.3.3 Resource Utilization
The resource utilization for mac_ui_edit_cont_sched is not a concern.

3.1.1.7.3.4 Program Library
Module mac_ui_DDS_edit_cont_sched is implemented as a stand-alone Oracle form.

3.1.1.7.4 mac_ui_gen_tape_label
Figure 3-17 shows the Generate Tape Label form. Figure 3-11 shows how this form is activated.

![Generate Tape Label](image)

**Figure 3-17 Generate Tape Label**

3.1.1.7.4.1 Purpose
The purpose of the mac_ui_gen_tape_label window is to provide operations with a graphical user interface for creating Backup/Archive tape labels for replacement of labels lost or damaged.

The System drop-down list will allow selection of the system id (DDS or one of the CTSs) where the tape was initialized into the DCS Backup/Archive. The values used in this drop-down list are defined as DCS_HW_STRING_ID in table DCS_CONFIGURATION and CAPT_SYS_ID in table DCS_CAPTURE_ACCT.

The Tape ID edit box will allow the operator to type in the tape identifier.
This module is derived from requirements delineated in paragraph 3.2.1.6.8 of the SRS.

3.1.1.7.4.2 Development Status
The mac_ui_gen_tape_label is reused without change.

3.1.1.7.4.3 Resource Utilization
The resource utilization for mac_ui_gen_tape_label is not a concern.

3.1.1.7.4.4 Program Library
Module mac_ui_gen_tape_label is implemented as a stand-alone Oracle form.

3.1.1.7.5 mac_ui_ingest_cont_sched
Figure 3-18 shows the Ingest Contact Schedules form. Figure 3-10 shows how this form is activated.
Note that before this form is activated, process mac_InsertFileNames is executed to populate table DCS_CONTACT_SCHED_FILES.

![Ingest Contact Schedules](image_url)

Figure 3-18 Ingest Contact Schedules

3.1.1.7.5.1 Purpose
The purpose of the mac_ui_ingest_cont_sched window is to provide operations with a graphical user interface for manually ingesting contact schedule files.

9/10/09 - 46 - DCS 104.4
The files listed are from the FILE_NAME field of table DCS_CONTACT_SCHED_FILES.

Requirements delineated in paragraph 3.2.1.1.4 of the SRS are allocated to the mac_ui_ingest_cont_sched.

3.1.1.7.5.2 Development Status

The mac_ui_ingest_cont_sched will be reused without change.

3.1.1.7.5.3 Resource Utilization

The resource utilization for mac_ui_ingest_cont_sched is not a concern.

3.1.1.7.5.4 Program Library

Module mac_ui_ingest_cont_sched is implemented as a stand-alone Oracle form.

3.1.1.7.6 mac_ui_manual_capt

Figure 3-19 Manual Data Capture shows the Manual Data Capture form. Figure 3-10 shows how this form is activated.
3.1.1.7.6.1 Purpose

The purpose of the mac_ui_manual_capt window is to provide operations with a graphical user interface for performing manual raw data captures.

The operator will be able to select which **Capture System** is to perform the capture (values are obtained from table DCS_CAPTURE_ACCT), the rawfile **Priority**, the **Mission ID** (values are obtained from table MISSION_ACCT), and the **Duration** of capture.

The stations in the **Station of Origin** drop-down list are defined in table MISSION_STATION_ACCT.

The **Station Downlink Times** represent the AOS and LOS times for the downlink at the station.
The **Recapture** checkbox allows the operator to enter duplicate **Start Time / Station / Capture System** (which defines the Capture Source). Without this setting, DCS will not allow duplicate information. With this setting, DCS will allow the information to be used for a re-capture.

The **Dual Downlink** checkbox allows the operator to enter duplicate **Start Time / Station / Capture System** (which normally defines the **Capture Source**). Without this setting, DCS will not allow duplicate information. With this setting, DCS will allow the information except that DCS will generate a unique **Capture Source** to be used for manual captures for dual downlinks.

The **Delete File** checkbox allows the operator to stop a manual capture and immediately delete the raw file (essential abort the capture).

The **Start** button executes the capture on the CTS, hides the **Start** button, and displays the **Stop** button.

The **Stop** button terminates the capture on the CTS, hides the **Stop** button, and displays the **Start** button.

The **Cancel** button closes the **Manual Data Capture** window with no action.

Requirements delineated in paragraph 3.2.1.1.6 of the SRS are allocated to the `mac_ui_start_capt`.

### 3.1.1.7.6.2 Development Status

The `mac_ui_manual_capt` window was updated from `mac_ui_start_capt` with the following changes:

- **CCR 1645 -- Dual Downlinks from Polar Tapes on the Same CTS**
  - the database is queried for previous captures that have the same Capture System, Mission Id, Station of Origin, and Start Time (year, day, and hour).
  - If duplicates exist, the operator is notified to use the **Recapture** or **Dual Downlink** buttons.

- **CCR 1660 -- START and STOP CAPTURE GUI on DCS_GUI to be combined to 1 function**
  - the GUI form name was changed to “Manual Data Capture” (the “Control” menu was changed from “Start Capture” to “Manual Capture”, “Stop Capture” on menu was removed).
  - the OK button was changed to START and the button does not close the window.
  - A STOP button was added to perform a normal stop capture (stop capture and transfer file).

- **CCR 1661 -- DCS manual capture "DO NOT TRANSFER" functionality changes.**
3.1.1.7.6.3 Resource Utilization

The resource utilization for mac_ui_start_capt is not a concern.

3.1.1.7.6.4 Program Library

Module mac_ui_start_capt is implemented as a stand-alone Oracle form.

3.1.1.7.7 mac_ui_start_copy_to_tape

Figure 3-20 shows the Start Copy to Tape form. Figure 3-10 shows how this form is activated.

![Start Copy to Tape](image)

**Figure 3-20 Start Copy to Tape**

3.1.1.7.7.1 Purpose

The purpose of the mac_ui_start_copy_to_tape window is to provide operations with a graphical user interface for performing Backup/Archive of selected raw data files.
The Select Tape Drive scrollbox displays available tape drives and the corresponding status.

If the Check Tape Date checkbox is checked the Backup/Archive function will be invoked with the check tape date option.

Requirements delineated in paragraph 3.2.1.6.8 of the SRS are allocated to the mac_ui_start_copy_to_tape.

3.1.1.7.7.2 Development Status

The mac_ui_start_copy_to_tape window will be reused without change.

3.1.1.7.7.3 Resource Utilization

The resource utilization for mac_ui_start_copy_to_tape is not a concern.

3.1.1.7.7.4 Program Library

Module mac_ui_start_copy_to_tape is implemented as a stand-alone Oracle form.

3.1.1.7.8 mac_ui_start_restage

Figure 3-21 shows the Start Restage form. Figure 3-10 shows how this form is activated.

![Start Restage Form]

**Figure 3-21 Start Restage**

3.1.1.7.8.1 Purpose

The purpose of the mac_ui_start_restage window is to provide operations with a graphical user interface that will identify the tape needed for restaging a particular raw data file from tape. The raw file will be selected on the main window, and then RESTAGE button clicked. When the button is clicked,
Start Restage window will appear with the filename and tape-id (value is obtained from table BACKUP_ACCT) displayed. 

**NOTE:** if the operator selects [Control->Ingest Tape] from the main menu, all of the files on the currently mounted tape will be ingested.

Requirements delineated in paragraph 3.2.1.6.9 of the SRS are allocated to the mac_ui_start_restage window.

### 3.1.1.7.8.2 Development Status

The mac_ui_start_restage window was reused without change. **Comment [bjp3]:** This form should have had tape device selection added.

### 3.1.1.7.8.3 Resource Utilization

The resource utilization for mac_ui_start_restage is not a concern.

### 3.1.1.7.8.4 Program Library

Module mac_ui_start_restage is implemented as a stand-alone Oracle form.

### 3.1.1.7.9 mac_ui_DDS_config

Figure 3-22 DDS Configuration GUI shows the DDS Configuration GUI form. Figure 3-10 shows how this form is activated.
3.1.1.7.9.1 Purpose

The purpose of the mac_ui_DDS_config window is to provide operations with a graphical user interface that will allow configuration of DDS runtime parameters. The GUI modifies table DCS_CONFIGURATION. Requirements delineated in paragraph 3.2.1.1 of the SRS are allocated to the mac_ui_DDS_config window.

3.1.1.7.9.2 Development Status

The mac_ui_DDS_config window was reused without change.

3.1.1.7.9.3 Resource Utilization

The resource utilization for mac_ui_DDS_config is not a concern.

3.1.1.7.9.4 Program Library

Module mac_ui_DDS_config is implemented as a stand-alone Oracle form.
### 3.1.1.7.10  mac_ui_mission_config

Figure 3-23 Mission Configuration GUI shows the Set Mission Parameters GUI form. Figure 3-10 shows how this form is activated.

![Set Mission Parameters GUI](image)

**Figure 3-23 Mission Configuration GUI**

### 3.1.1.7.10.1 Purpose

The purpose of the mac_ui_mission_config window is to provide operations with a graphical user interface that will allow creation and configuration of DCS supported missions. Items that can be set from this form include:

- **Mission Identifier** – unique identifier for a mission (see Table 1: MISSION_ACCT).
- **Mission Name** – full name of the mission (see Table 1: MISSION_ACCT).
- **Data Type** – unique identifier for data type (see Table 1: MISSION_ACCT).
- **Data Type Description** – description of data type.
- **Default Priority** – default priority for mission (see Table 1: MISSION_ACCT).
- **Raw Data Path** – DDS directory for mission raw data (see Table 1: MISSION_ACCT).
- **Station Id(s)** – ground station identifiers for mission (see Table 2: MISSION_STATION_ACCT).
• **Capture System(s)** – Capture/Transfer Subsystem identifiers for the mission (see Table 7: CAPTURE_MISSION_ACCT).
• **Destination System(s)** – destination system identifiers for the mission (see Table 4: ROUTING_ACCT).
• **Initial Xfer Status** – the initial value for Xfer_Status.

Requirements delineated in paragraph 3.1.2.1 of the SRS are allocated to the mac_ui_mission_config window.

3.1.1.7.10.2 Development Status

The mac_ui_mission_config window was updated for CCR 1646 & CCR 1904.

3.1.1.7.10.3 Resource Utilization

The resource utilization for mac_ui_mission_config is not a concern.

3.1.1.7.10.4 Program Library

Module mac_ui_mission_config is implemented as a stand-alone Oracle form.

3.1.1.7.11 mac_ui_dest_config

Figure 3-24 shows the Destination System Configuration GUI form. Figure 3-10 shows how this form is activated.

![Destination System Identification](image)

**Figure 3-24 Destination System Configuration GUI**

3.1.1.7.11.1 Purpose

The purpose of the mac_ui_dest_config window is to provide operations with a graphical user interface that will allow creation and configuration of DCS supported mission-defined raw data destination system.
The GUI modifies values in table DESTINATION_ACCT. Requirements delineated in paragraph 3.1.2.2 of the SRS are allocated to the mac_ui_dest_config window.

3.1.1.7.11.2 Development Status
The mac_ui_dest_config window was reused without change.

3.1.1.7.11.3 Resource Utilization
The resource utilization for mac_ui_dest_config is not a concern.

3.1.1.7.11.4 Program Library
Module mac_ui_dest_config is implemented as a stand-alone Oracle form.

3.1.1.7.12 mac_ui_rawfile_details
Figure 3-25 shows the Raw File Details GUI form. Figure 3-10 shows how this form is activated.
3.1.1.7.12.1 Purpose

The purpose of the mac_ui_rawfile_detail window is to provide operations with a graphical user interface that will allow reviewing/updating raw file detailed information.

Fields that are not modifiable are shown in gray. Values are obtained from table DCS_RAWFILE_ACCT, BACKUP_ACCT, and TRANSFER_ACCT.

Requirements delineated in paragraph 3.1.1.5, 3.1.2.4, and 3.1.2.5 of the SRS are allocated to the mac_ui_rawfile_detail window.

3.1.1.7.12.2 Development Status

The mac_ui_rawfile_detail window was updated as follows:
- CCR 1943 – **Ready, Hold**, and **Queue** buttons are enabled even when the **Status** is ‘PROGRESS’ if the user is DCS_MANAGER. This allows DCS operations managers to ‘take back’ control of a file that a destination system has taken control of and subsequently cannot handle.

3.1.1.7.12.3 Resource Utilization

The resource utilization for mac_ui_rawfile_detail is not a concern.

3.1.1.7.12.4 Program Library

Module mac_ui_rawfile_detail is implemented as a stand-alone Oracle form.

3.1.1.7.13 mac_ui_xfer_history

Figure 3-25 shows the Transfer History GUI form. Figure 3-10 shows how this form is activated.

![Transfer History GUI](image)

**Figure 3-26 Transfer History GUI**

3.1.1.7.13.1 Purpose

The purpose of the mac_ui_xfer_history window is to provide operations with a graphical user interface that will allow reviewing previous transfer success and failures for a file to a particular destination.

Values are obtained from table TRANSFER_ACCT_ARCHIVE.

Requirements delineated in paragraph 3.1.1.5, 3.1.2.4, and 3.1.2.5 of the SRS are allocated to the mac_ui_rawfile_detail window.
3.1.1.7.13.2 Development Status

The mac_ui_xfer_history window is new for DCS 1.2.

3.1.1.7.13.3 Resource Utilization

The resource utilization for mac_ui_xfer_history is not a concern.

3.1.1.7.13.4 Program Library

3.1.1.7.14 mac_ui_trans_cts_file

Figure 3-27 shows the Transfer CTS File GUI form. Figure 3-10 shows how this form is activated.

![Transfer CTS File GUI](image)

**Figure 3-27 Transfer File GUI**

3.1.1.7.14.1 Purpose

The purpose of the mac_ui_trans_file window is to provide operations with a graphical user interface that allows selecting one or more raw files for transfer from a selected CTS.

Values are obtained from table TEMP_CTS_FILES.
Requirements delineated in paragraph 3.2.1.12 of the SRS are allocated to the mac_ui_trans_cts_file window.

3.1.1.7.14 Development Status
The mac_ui_trans_cts_file window is new for DCS 1.2.

3.1.1.7.14.3 Resource Utilization
The resource utilization for mac_ui_trans_cts_file is not a concern.

3.1.1.7.14.4 Program Library
Module mac_ui_trans_cts_file will be implemented as a stand-alone Oracle form.

3.1.1.7.15 mac_ui_init_tape
Figure 3-29 shows the Initialize Tape GUI. Figure 3-11 shows how this form is activated.

Figure 3-28 Initialize Tape
3.1.1.7.15.1 Purpose

The purpose of the mac_ui_initialize_tape window is to provide operations with a graphical user interface for initializing Backup/Archive tapes (primarily to re-use outside tapes within the DCS archive).

The DDS and CTS radio buttons and CTS system drop-down list allow selection of the system id (DDS or one of the CTSs) where the tape will be initialized into the DCS Backup/Archive. The values used in this drop-down list are defined as DCS_HW_STRING_ID in table DCS_CONFIGURATION and CAPT_SYS_ID in table DCS_CAPTURE_ACCT.

The Select Tape Drive edit boxes will allow the operator to view the tape drive status and select the drive to perform the initialization.

This module is derived from requirements delineated in paragraph 3.2.1.6.8 of the SRS.

3.1.1.7.15.2 Development Status

The mac_ui_init_tape has been updated for DCS 1.3 with the following change:

- CCR 2067 – DCS Initialize Tape GUI Refresh Problem.
  An ampersand was added to the command string to initialize tapes in the background on the DDS.

3.1.1.7.15.3 Resource Utilization

The resource utilization for mac_ui_init_tape is not a concern.

3.1.1.7.15.4 Program Library

Module mac_ui_init_tape is implemented as a stand-alone Oracle form.

3.1.2 Capture Transfer Subsystem (CTS)

The Capture Transfer Subsystem (CTS) is responsible for capturing the wideband data to disk and transferring the raw file to the Data Capture System (DCS) Database Server (DDS). Figure 3-29 shows the software units that comprise the CTS.
Normally, operations will control CTS operations remotely from the DDS (see section 3.3.1.1). As configuration parameters or schedules change, the relevant files are sent to the CTS subsystem(s). After the file is delivered, a signal is sent from the DDS to the rdc_AutoCapture daemon process to carry out the action.

### 3.1.2.1 AutoCapture Component

The rdc_AutoCapture process runs as a daemon process on the Capture Transfer Subsystem. The purpose of the rdc_AutoCapture process is to automatically start the rdc_XXXXX processes at appropriate times.

All of the requirements delineated in paragraph 3.2.2.1 of the SRS are allocated to the rdc_AutoCapture process.
3.1.2.1.1 Purpose

Module rdc_AutoCapture will be the main function responsible for automatically running the raw data capture processes as well as initiating auto file transfer via “rdc_AutoTransfer”.

3.1.2.1.2 Development Status

rdc_AutoCapture was updated with the following changes:

- CCR 1645 -- Dual Downlinks from Polar Tapes on the Same CTS
  - An optional “channel id” will be added to the manual capture parameters file.
  - The optional “channel id” will be passed on the command line to rdc_Capture.

- CCR 1721 -- CTS1 failed to start. CTS2 started late.
  - “rdc_AutoTransfer” was created to transfer data independently of performing AutoCaptures (note: “rdc_AutoTransfer” is really a forked child process of rdc_AutoCapture, not a separate process).

- CCR 2170 -- DCS - LCTS2 Capture failure with unable to open device error.
  - “rdc_CaptureIsRunning” will be updated to use a ‘ps’ command whose output format does not change over time (e.g. after 24hours) and thus will always correctly identify a currently running process.

3.1.2.1.3 Resource Utilization

The resource utilization for rdc_AutoCapture.c is not a concern.

3.1.2.1.4 Program Library

Module rdc_AutoCapture is implemented as a stand-alone daemon process.

3.1.2.2 Raw Data Capture Component

3.1.2.2.1 Purpose

The rdc_Capture process will run either from the rdc_AutoCapture daemon process or from the command line. The purpose of the rdc_Capture process is to capture one 75 Megabits per Second data stream to disk and create an associated accounting file.

All of the requirements delineated in paragraph 3.2.2.2 of the SRS are allocated to the rdc_Capture process.

3.1.2.2.2 Development Status

The rdc_Capture was updated as follows:
• CCR 1645 -- Dual Downlinks from Polar Tapes on the Same CTS
  o An optional "channel id" will be added to the command line (passed from rdc_AutoCapture).
  o If the "channel id" is not present, the "channel id" will be derived using the current algorithm.

• CCR 1653 -- File Size Problem in Accounting File causing problems transferring to LPSNG
  o The 2 second sleep for DMA completion will be updated to 10 seconds (to allow AIO completion).
  o The asynchronous I/O buffer queue will be "sync'd" before closing the file.
  o The file will be closed before calling stat() to get the filesize.

• CCR 1661 -- DCS manual capture "DO NOT TRANSFER" functionality changes.
  o Accept signal SIGHUP as "stop capture and delete file"

• CCR 2006 -- DCS -- FILE_SIZE field in DB does not work for ASN data.
  o rdc_Capture processes will be updated to use an integer number of bytes for file size (.acct files will still use MB for backward compatibility).

3.1.2.3 Resource Utilization

The rdc_Capture will run on an isolated central processing unit (CPU). Since the CTS machines only have two CPUs and one cannot be isolated, the capture processes will need to share one CPU.

Each capture process allocates a ring buffer of twenty-five 4MB buffers (100MB) for asynchronous writing to disk. Plus, each process allocates two 4MB buffers for DMA transfers from the Myriad card.

Also, other resources in use during capture could interfere with capture (e.g. disk accesses, other processes which limit I/O throughput).

3.1.2.4 Program Library

Module rdc_Capture is implemented as a stand-alone process.

3.1.2.3 Transfer To Tape Component

3.1.2.3.1 Purpose

The rdc_Save process runs either from the rdc_AutoTransfer daemon process when enabled or from the command line. The rdc_InitTape process runs from the command line.
The purpose of the rdc_Save process is to write a raw data file to tape. All of the requirements delineated in paragraphs 3.2.2.3 of the SRS are allocated to the rdc_Save process. The purpose of module rdc_Save is to copy the captured raw wideband data file to tape. The purpose of the rdc_InitTape process is to initialize tapes for use in the DCS Backup/Archive.

3.1.2.3.2 Development Status

Module rdc_Save will be modified from existing source, with the following changes:

- CCR 1653 -- File Size Problem in Accounting File causing problems transferring to LPSNG
  - The .acct file info (e.g. file_size) will be validated before copying to tape.
- CCR 1884 -- DCS - Remove dependency of accounting file for ASA data
  - The tape device is passed in on the command line instead of using an environment variable.
  - A common routine is used to read the .acct file info.
- CCR 1619 -- CTS can only use tapes that are new or have been reinitialized.
  - Call new init tape routine for new tapes.

Module rdc_InitTape was created from existing rdc_Save source, based on the following CCR:

- CCR 1619 -- CTS can only use tapes that are new or have been reinitialized.
  - The module creates a tapeHeader file and writes it to the beginning of the tape.

3.1.2.3.3 Resource Utilization

The resource utilization for rdc_Save and rdc_InitTape is not a concern.

3.1.2.3.4 Program Library

Modules rdc_Save and rdc_InitTape are implemented as a stand-alone process.

3.1.2.4 Generate Label Component

3.1.2.4.1 Purpose

The rdc_GenStackedLabel process will run either from the rdc_Save process or from the command line.

The purpose of the rdc_GenStackedLabel process is to create a tape label. All of the requirements delineated in paragraph 3.2.1.2.10.3 of the SRS are allocated to the rdc_GenStackedLabel process.
3.1.2.4.2 Development Status
Module rdc_GenStackedLabel will be reused without change.

3.1.2.4.3 Resource Utilization
The resource utilization for rdc_GenStackedLabel is not a concern.

3.1.2.4.4 Program Library
Module rdc_GenStackedLabel.c will be implemented as a stand-alone process.

3.1.2.5 Delete Raw Files Component (CTS)

3.1.2.5.1 Purpose
The rdc_DeleteFiles process will run either from the rdc_AutoTransfer process or from the command line.

The purpose of the rdc_DeleteFiles process is to remove raw files from on-line storage. All of the requirements delineated in paragraph 3.2.2.4 of the SRS are allocated to the rdc_DeleteFiles process.

The purpose of module rdc_DeleteFiles.c is to delete the RDC capture data file and the associated accounting file. The file protection is checked. If the file is not read-only, it is deleted. If the file is read-only an error message is logged and the file is not deleted.

3.1.2.5.2 Development Status
Module rdc_DeleteFiles.c will be reused without change.

3.1.2.5.3 Resource Utilization
The resource utilization for rdc_DeleteFiles is not a concern.

3.1.2.5.4 Program Library
Module rdc_DeleteFiles will be implemented as a stand-alone process.

3.1.2.6 Raw File Transfer Component (CTS)

3.1.2.6.1 Purpose
The purpose of module rdc_TransferFile is to transfer the raw data file and the associated accounting file to the DDS. After successful transfer, the file protection is removed.

3.1.2.6.2 Development Status
Module rdc_TransferFile will be updated with the following changes:
- CCR 1653 -- File Size Problem in Accounting File causing problems transferring to LPSNG
  - The .acct file info (e.g. file_size) will be validated before sending to DDS.

3.1.2.6.3 Resource Utilization

The resource utilization for rdc_TransferFile.c is not a concern.

3.1.2.6.4 Program Library

Module rdc_TransferFile.c will be implemented as a stand-alone process.

3.1.2.7 Raw Data Transmit Component

3.1.2.7.1 Purpose

The purpose of module rdc_Transmit.c is to transmit a data file out the Myriad Logic capture card for test purposes.

3.1.2.7.2 Development Status

Module rdc_Transmit will be reused without change.

3.1.2.7.3 Resource Utilization

The resource utilization for rdc_Transmit is not a concern.

3.1.2.7.4 Program Library

Module rdc_Transmit is implemented as a stand-alone process.

3.1.2.8 Journaling Component

The Journaling Component on the CTS will reuse the journaling component on the DDS.

3.1.2.9 Moving Window Display (MWD) Formatter Component

3.1.2.9.1 Purpose

The purpose of the MWD Formatter Component is to frame sync, format, and subsample incoming raw wideband data and deliver the subsampled imagery to the Capture Transfer Display subsystem’s MWD.

3.1.2.9.2 Development Status

MWD Formatter Component will be reused from previous projects and will require numerous changes.
3.1.2.9.3 Resource Utilization

The resource utilization for the MWD Formatter could become significant.

3.1.2.9.4 Program Library

The MWD Formatter Component will be implemented as a stand-alone process.

3.1.3 Capture Transfer Display (CTD)

The Capture Transfer Display (CTD) subsystem is responsible for creating and showing a Moving Window Display for the incoming wideband data. It will host the MWD server component.

3.2 Concept of execution

3.2.1 Normal Raw Data Flow

Refer to Figure 3-29 for raw data flow through the CTS. Normal Raw Data Flow begins with the rdc_Capture process. The rdc_AutoCapture process will start an rdc_Capture process to retrieve data from each Myriad Logic capture card device and save it to disk. rdc_AutoCapture also starts a MWDFormatter process.

The rdc_Capture processes provide data to the MWDFormatter via a shared memory ring buffers. At the end of capture, each rdc_Capture will create an Accounting File for the raw file it received and return with an exit status showing successful capture.

When all rdc_Capture processes have finished, rdc_AutoCapture will start rdc_AutoTransfer for each raw data file to transfer the file to the DDS. After all transfers are complete, rdc_AutoTransfer will start rdc_Delete to remove the local files.

rdc_Transfer will use ftp to transfer each raw data file (and accounting file) to the DDS. After each transfer, rdc_Transfer will execute mac_UpdDCSAcct on the DDS (via the remote shell, 'rsh').

Refer to Figure 3-9 for raw data flow on the DDS. mac_UpdDCSAcct will read the accounting file, create a record in table DCS_RAWFILE_ACCT, and create transfer record(s) in table TRANSFER_ACCT with the XFER_STATUS set to 'IN-QUEUE'. The transfer records are created based on the MISSION_ID recorded in the accounting file and the raw file routing information setup in table ROUTING_ACCT.

DDS automation will select the oldest, highest priority file from those that are 'IN-QUEUE' for each destination and set it to 'READY'.

When each destination system sees the 'READY' status for their destination id, it will update their corresponding XFER_STATUS to 'PROGRESS' and ftp fetch...
the raw file. After the transfer is complete (i.e. when the destination system no longer needs the file from DDS), the destination will set the XFER_STATUS to ‘SUCCESS’. If the destination system fails to retrieve the file, it will set the XFER_STATUS field to a failure code.

3.3 Interface design.

3.3.1 Interface identification and diagrams.

3.3.1.1 DDS to CTS Interface

The DDS to CTS Interface is used to pass data and information between the DDS and each CTS. Both the DDS and the CTS are being updated, so the interface has specific requirements imposed on it. Refer to Figure 3-9, Figure 3-4, Figure 3-5, Figure 3-6, and Figure 3-29 for diagrams depicting portions of this interface.

3.3.1.2 DDS to Destination Systems Interface

The DDS to Destination Systems Interface is used to communicate raw data delivery between the DDS and each destination system retrieving raw data from the DDS. The DDS is being updated without changes to this interface, so the interface imposes requirements on the DDS. Refer to the DCS Database Schema in Figure 3-2.

3.3.1.3 MOC to DDS Interface

The MOC to DDS Interface is used to transfer Contact Schedules from the Mission Operations Center(s) to the DDS. Only the DDS is being modified for this effort, so the interface imposes specific requirements on the DDS. This interface is depicted in Figure 3-4.

3.3.1.4 RDC to MWD Formatter Interface

The RDC to MWD Formatter Interface is used to transfer raw wideband data to the MWD Formatter Component for formatting and displaying. This interface is depicted in Figure 3-29.

3.3.1.5 CTS to CTD Interface

The CTS to CTD Interface is used to transfer formatted and subsampled video imagery and other information from the CTS to the Capture Transfer Display subsystem. This interface is depicted in Figure 3-29.

3.3.2 DDS to CTS Interface

The DDS to CTS Interface is used to pass CTS Setup Information, Manual Raw Data Capture Parameters and CTS Support Schedule information from the DDS to the CTS(s). The interface is also used to electronically transfer captured Raw Data files from each CTS to the DDS.
3.3.2.1 Priority Assigned

The operator initiates changes to and delivery of CTS Setup Information and Manual Raw Data Capture Parameters on the DDS. Changes to and delivery of the CTS Support Schedule are initiated by the DDS whenever a mission contact schedule is ingested (usually from automatic polling) or modified (usually from operator updates). Receipt of Raw Data from the CTS is initiated by the CTS. These actions are performed independent of other DDS operations and thus will run at normal system priority.

Receipt of CTS Setup Information, Manual Raw Data Capture Parameters and CTS Support Schedule on the CTS is initiated by the DDS signaling the CTS. Ingest of the new information will occur immediately.

Delivery of Raw Data from CTS to DDS is initiated automatically, immediately after capture. The CTS will initiate ingest of the new raw data file on the DDS.

3.3.2.2 Type of Interface

The DDS to CTS Interface is in the form of data files delivered via File Transfer Protocol and remote commands executed via remote shell (rsh). The rsh command will be a ‘kill’ command to send a signal or, in the case of Raw Data delivery, execute DDS process mac_UpdDCSAcct to register the data in the database and signal destination systems of available data.

3.3.2.3 Characteristics of Data Elements

3.3.2.3.1 CTS Setup Information

CTS Setup Information is an ASCII text file that provides operating parameters and options for the CTS. Each parameter appears on a separate line and is preceded by an identifying field name and an equals sign, e.g. “<field name> = value”.

3.3.2.3.1.1 Individual Data Elements

Mission_Identification – character string of 1 to 5 characters that uniquely identify the default mission (note these values will be defined in table MISSION_ACCT on the DDS).

Transfer_Method – character string set to one of the following:
  FTP indicates the CTS will ftp transfer raw files to the DDS.
  TAPE indicates the CTS will tar the raw files to tape.
  NONE indicates the CTS will not transfer the raw files (they will remain on the CTS).

Transfer_Idle_Time – integer indicating number of hours of idle (no scheduled captures) time needed before beginning raw file transfer.

FTP_Address – character string set to the internet address of the DDS (used only if the Transfer Method is ‘FTP’).
FTP_Directory – character string set to the directory/path on the DDS to receive raw files (used only if the Transfer Method is ‘FTP’).

FTP_User_Name – character string set to the DDS user name for ftp login (used only if the Transfer Method is ‘FTP’).

FTP_Password – character string set to the DDS user name password for ftp login (used only if the Transfer Method is ‘FTP’).

Delete_After_Transfer – character string set to one of the following:
   YES indicates the CTS will delete the raw files after successful transfer to the DDS or tape.
   NO indicates the CTS will not delete the raw files after successful transfer to the DDS or tape.

Num_Capture_Systems – The number of CTS systems at this particular ground station. This parameter is used when generating a unique “capture source” if the CTS is supporting two Myriad cards. The “capture source” for the second card is the same as the first plus this value.

Tape_Device – The tape device name to use for saving files to tape.

MWD_Name – The MWD system id to receive the scrolling imagery.

/hw/myriad/<N> – The channel assignment for slot <N>.

Example:
   Mission_Identification= L7ET
   Transfer_Method= FTP
   Transfer_Idle_Time= 2
   FTP_Address= dds001
   FTP_Directory= /u01/ops/rawdata
   FTP_User_Name= opsftp
   FTP_Password= pass
   Delete_After_Transfer= YES
   Num_Capture_Systems= 3
   Tape_Device= /hw/tape/tps1d4nrsnv
   MWD_Name= 2
   /hw/myriad/6 = 3

3.3.2.3.1.2 Sources and Recipients

On the DDS, module mac_GenSetup will create a CTS Setup Information file based on records in table DCS_CAPTURE_ACCT, then ftp the file to the CTS and send a “New Setup” signal to rdc_AutoCapture. Figure 3-5 shows this portion of the interface.

On the CTS, module rdc_AutoCapture will receive the “New Setup” signal and re-read the CTS Setup Information file. Figure 3-29 shows this portion of the interface.

The requirements of paragraph 3.1.1.2 are allocated to this portion of the design.
3.3.2.3.2 Manual Capture Parameters

Manual Capture Parameters are delivered in an ASCII text file that provides parameters and options for the CTS to perform a manual capture (i.e. capture initiated by an operator). Each parameter appears on a separate line and is preceded by an identifying field name and an equals sign, e.g. “<field name> = value”. Manual captures are generally used to capture data from wideband tapes.

3.3.2.3.2.1 Individual Data Elements

Mission_Identification – character string of 1 to 5 characters that uniquely identify the mission (note these values will be defined in table MISSION_ACCT on the DDS).

Original_Station – character string of 1 to 5 characters that uniquely identify the receiving ground station that downlinked the data and recorded it to wideband tape.

Scheduled_Start_Time – character string of 11 characters that identify the starting date/time of the satellite downlink (format: YYDDDHHMMSS).

Scheduled_Stop_Time – character string of 11 characters that identify the ending date/time of the satellite downlink (format: YYDDDHHMMSS).

Priority – integer from 0 to 9.

Source – integer from 0 to 9, or ‘*’, optional field. If not present or set to ‘*’, the channel is derived from the Lcts hostname and the Myriad Logic capture card slot number.

Example:

Mission_Identification= L7ET
Original_Station= AGS
Scheduled_Start_Time= 00094023442
Scheduled_Stop_Time= 00094024411
Priority= 1
Source= 3

3.3.2.3.2.2 Sources and Recipients

On the DDS, module mac_ManCapture will create a Manual Capture Parameter File based on information entered by the operator in the GUI (see Figure 3-19), then ftp the file to the selected CTS and send a “Start Manual Capture” signal to the rdc_AutoCapture process.

Figure 3-6 shows this portion of the interface.

On the CTS, module rdc_AutoCapture will receive the “Start Manual Capture” signal, read the Manual Capture Parameters file, and perform the capture.

Figure 3-29 shows this portion of the interface.

The requirements of paragraph 3.1.1.3 are allocated to this portion of the design.
3.3.2.3.3 CTS Support Schedule

The CTS Support Schedule is delivered as an ASCII text file that provides parameters and options for the CTS to perform automated captures. All parameter values for a scheduled capture appear on a single line. This file does not include field names or equal signs.

3.3.2.3.3.1 Individual Data Elements

Mission_Identification – character string of 1 to 5 characters that uniquely identify the mission (note these values will be defined in table MISSION_ACCT on the DDS).

Scheduled_Start_Time – character string of 11 characters that identify the starting date/time of the downlink (format: YYDDDHHMMSS).

Scheduled_Stop_Time – character string of 11 characters that identify the ending date/time of the downlink (format: YYDDDHHMMSS).

Priority – integer from 0 to 9.

Example:
L7 00-094-02:34:42 00-094-02:44:11 1
L7 00-094-04:13:42 00-094-04:18:11 2

3.3.2.3.3.2 Sources and Recipients

On the DDS, module mac_GenSuptSchedule will create a Support Schedule File, based on contact information in the CONTACT_SCHEDULE database table, then ftp the file to CTS and send a “New Schedule” signal to the rdc_AutoCapture process running on the CTS. Figure 3-4 shows this portion of the interface.

On the CTS, module rdc_AutoCapture will receive the “New Schedule” signal and re-read the Support Schedule file. Figure 3-29 shows this portion of the interface.

The requirements of paragraph 3.1.1.4 are allocated to this portion of the design.

3.3.2.3.4 Raw Data

Raw Data is delivered from the CTS to the DDS as two files: an ASCII text file that provides accounting information about the raw file and a binary data file that contains the raw captured data. Each accounting parameter appears on a separate line and is preceded by an identifying field name and an equals sign, e.g. “<field name> = value”.

3.3.2.3.4.1 Individual Data Elements

CTS_Capture_String – character string of 1 to 10 characters that uniquely identifies the CTS system the wideband data was captured on (note these values will be defined in the DCS_CAPTURE_ACCT).
Capture_Source – character string of 1 to 3 characters that identifies the ground station data source delivering wideband data to the CTS.

Original_Station – character string of 1 to 5 characters that uniquely identifies the original receiving ground station that downlinked the wideband data.

Scheduled_Start_Time – character string of 11 characters that identifies the starting date/time of the satellite downlink (format: YYDDDHHMMSS).

Scheduled_Stop_Time – character string of 11 characters that identifies the ending date/time of the satellite downlink (format: YYDDDHHMMSS).

Actual_Start_Time – character string of 11 characters that identifies the actual starting date/time of the CTS capture (format: YYDDDHHMMSS).

Actual_Stop_Time – character string of 11 characters that identifies the actual ending date/time of the CTS capture (format: YYDDDHHMMSS).

Raw_Data_File_Name – character string of 1 to 256 characters that identifies the raw data file name.

File_Size – fixed-point number that represents the number of megabytes in the file (the number of decimal places is 6, so the actual number of bytes can be determined).

Received_Data_Vol – number of megabytes of data received through the capture card channel (should be the same as File_Size unless a write error occurred writing to the disk file).

Expected_Data_Vol – number of megabytes of data expected. This number is estimated from the actual start of data transmission through the end of the capture session, which is determined by the downlink times.

Scheduled_Data_Vol – number of megabytes of data scheduled. This number is estimated from the scheduled downlink start time through the scheduled downlink end time, which makes it the same as Expected_Data_Vol.

Transmission_Rate – reception rate of the wideband data (in megabytes per second). This number is the Received_Data_Vol divided by the overall capture session time (from downlink start through stop).

Isolate_Flag – integer (0 or 1) indicating if the capture process was executed on an isolated processor.

Suspend_Flag – integer (0 or 1) indicating if non-essential processes (e.g. ‘tar’ and ‘ftp’) that may interfere with the capture process were suspended during the capture.

Mission_Id – character string of 1 to 5 characters that uniquely identify the mission (note these values will be defined in table MISSION_ACCT on the DDS).

Priority – integer from 0 to 9.

Example:

```
CTS_Capture_String= cts001
```
3.3.2.3.4.2 Sources and Recipients

On the CTS, module rdc_Transfer will ftp the raw data file and its associated accounting file to the DDS and execute the mac_UpdDCSAcct process running on the DDS. Figure 3-29 shows this portion of the interface.

On the DDS, module mac_UpdDCSAcct will read the associated accounting file and update the database. Figure 3-9 shows this portion of the interface.

The requirements of paragraph 3.1.1.5 are allocated to this portion of the design.

3.3.2.3.5 Mission Information

The Mission Information file is delivered as an ASCII text file that provides the mission-specific information needed to perform captures. All parameter values for a mission id/channel appear on a single line. This file does not include field names or equal signs.

3.3.2.3.5.1 Individual Data Elements

**Mission_Identification** – character string of 1 to 5 characters that uniquely identify the mission (note these values will be defined in table MISSION_ACCT on the DDS).

**Data_Type** – 3-character field indicating the type of data.

**Bit_Rate** – character string of up to 11 numbers that identifies the data bit rate in bits-per-second.

**Channel_Id** – single character indicating which capture device channel will receive data. This id will be ‘a’, ‘b’, or ‘*’ for both.

Example:

```
L7ET NOM 75000000 *
L5ET NOM 85000000 a
```
3.3.2.3.5.2 Sources and Recipients

On the DDS, module mac_SendSetup will create a Mission Information file, based on information in the MISSION_ACCT database table, then ftp the file to CTS. Figure 3-5 mac_SendSetup shows this portion of the interface.

The requirements of paragraph 3.1.1.6 are allocated to this portion of the design.

3.3.3 DCS to Destination Systems Interface

The DCS to Destination Systems Interface is used to notify each destination system of available data and track the transfer of the data to the destination. Refer to the DCS to Destination Systems ICD (reference document #0) for complete details.

3.3.4 MOC to DDS Interface

The MOC to DDS Interface is used to transfer Contact Schedules from the Mission Operations Center to the DDS. The mission Contact Schedule is delivered as an ASCII text file that provides parameters and options for the DCS to perform automated captures. Several parameter values for a scheduled capture appear on a single line. This file does not include field names or equal signs.

This interface is depicted in Figure 3-4.

3.3.4.1 Contact Schedule

The contact schedule contains the following for each scheduled transmission of wideband and narrowband data to EDC:

- Predicted acquisition of signal (AOS) and loss of signal (LOS) times
- Scheduled start and stop times of data transmission (for wideband data only)
- X-band frequencies (for wideband data only)

The MOC produces the contact schedule daily and delivers it to the DDS before the first contact listed in the schedule. The contact schedule covers a 48-hour period. A revised contact schedule is sent whenever an unexpected schedule change occurs.

The Contact Schedule file format is defined in section 4.2.2 of the MOC to LGS ICD (see additional reference document #0).

Requirements defined in SRS 3.1.3.1 are allocated to this interface.

3.3.5 RDC to MWD Formatter Interface

The RDC to MWD Formatter Interface is used to transfer raw data buffers from the Raw Data Capture component to the Moving Window Display Formatter component.

This interface is depicted in Figure 3-29.
3.3.5.1 Priority Assigned
The raw data capture component will not wait for a free buffer. If the ring buffer is full, the current buffer will be discarded.

3.3.5.2 Type of Interface
Raw data is delivered via a ring buffer using POSIX compatible shared memory and counting semaphores.

3.3.5.3 Characteristics of Data Elements
3.3.5.3.1 rdc_mwd_shm_ctl
A portion of the shared memory defines the characteristics and control of the entire shared memory block.

3.3.5.3.1.1 Individual Data Elements
BufCmd – 64-bit integer used for sending commands & status from RDC to the MWD Formatter Ingest: 0=shared memory not ready (default); 1=shared memory is ready, begin reading; -1=end of contact, flush memory and stop.
BufSize – 64-bit integer indicating the number of bytes in each ring buffer block.
NumBufs – 64-bit integer indicating the number of ring buffer blocks.
FullSem – type sem_t, counting integer indicating the number of full buffers.
EmptySem – type sem_t, counting integer indicating the number of empty buffers.

3.3.5.3.1.2 Sources and Recipients
The rdc_Capture process is responsible for initializing the rdc_mwd_shm_ctl block. rdc_Capture will do the following:
1. set BufCmd to 0 (0 is the default when the shared memory is created).
2. set the BufSize and NumBufs values.
3. initialize the FullSem semaphore to 0 (indicating none are full).
4. initialize the EmptySem semaphore to NumBufs (indicating all are empty).
5. after initialization, the BufCmd will be set to 1, (indicating that the MWD Formatter Ingest can begin using the shared memory).

During capture, each time rdc_Capture receives an incoming data buffer, it will 'test and wait' the EmptySem to see if the MWDFormatter has not fallen too far behind. If the 'test and wait' shows that the MWDFormatter has not fallen behind, rdc_Capture will post the FullSem, otherwise it will not.

At the end of the contact, rdc_Capture will write a –1 into the BufCmd.

Upon start-up, the MWDFormatter Ingest routine will wait for BufCmd to become 1. Then use the BufSize and NumBufs values for appropriate initialization.
During execution, each time the MWDFormatter ingest needs another data buffer, it will 'test and wait' the FullSem to see if there's an available buffer. If one is available, the buffer is read from the next available buffer and processed. If none are available, the BufCmd is checked to see if end-of-contact has been reached. At the end of the contact, the MWD Formatter will shutdown.
3.3.5.3.2 Shared Arrays

Three shared blocks are also stored in the shared memory.

3.3.5.3.2.1 Individual Data Elements

**BufNumber** – array of 64-bit integers used to associate each raw data buffer in the ring with a sequence number. The MWD Formatter Ingest routine can use these numbers for “drop-out” detection. The array is **NumBufs** long.

**RingBuffers** – array of buffers holding the raw data. Each Buffer is **BufSize** bytes long and the array is **NumBufs** long.

**InsertTimes** – array of 64-bit integers holding the timestamps of each buffer written to the ring. These timestamps are useful for measuring throughput performance. The array is **NumBufs** long.

3.3.5.3.2.2 Sources and Recipients

Each time rdc_Capture receives an incoming data buffer it will increment an internal **buffer_number**. If a buffer is available, the buffer is written into the next slot in the **RingBuffers** array, the **buffer_number** is stored in the corresponding position of the **BufNumber** array, and the current time is stored in the corresponding position of the **InsertTimes** array.

Each time the MWD Formatter Ingest reads another data buffer, it will compare the associated value from the **BufNumbers** array with an internal **last_buffer_number** (for drop-out detection).

3.3.6 CTS to CTD Interface

The CTS to CTD Interface is used to transfer formatted and subsampled video imagery and other information from the CTS to the Capture Transfer Display subsystem.

3.3.6.1 Priority Assigned

This interface will use normal priorities for communication.

3.3.6.2 Type of Interface

The CTS to CTD Interface is in the form of commands and data delivered via Terminal Control Protocol / Internet Protocol (TCP/IP) socket communication.

3.3.6.3 Characteristics of Data Elements

3.3.6.3.1 MWD Commands

The interface will be in the form of socket messages carrying various MWD commands.
3.3.6.3.1.1 Individual Data Elements

**Command** – character string set to one of the following:
- **Init** commands the MWD to initialize / re-initialize.
- **Video** sends a single line of video for a single band of data to the MWD for display.
- **Text** sends a single line of text to the MWD for display to the right side of the video.
- **Interval** commands the MWD to start a new interval of data without reinitializing the MWD.
- **Gap** commands the MWD to start a new interval of data and inserts black video along with annotating text.

3.3.6.3.1.2 Sources and Recipients

On the CTS, the MWDFormatter module will create the MWD commands and write them to the socket. On the CTD the MWD server application will read the MWD commands from the socket and perform the requested function. Figure 3-29 shows this portion of the interface.

The requirements of paragraph 3.1.5.1 are allocated to this portion of the design.
Section 4  CSCI Detailed Design

4.1  DCS Database Server

4.1.1  Database Component

The details of the database component are given in the DCS Database Analysis & Design (applicable document #0).

4.1.2  Monitor and Control Component

4.1.2.1  mac_DDS

4.1.2.1.1  Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.1.2.1.2  Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.2.1.3  Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.1.2.1.4  Procedural Commands

mac_DDS will use Pro C embedded SQL to query table DCS_CONFIGURATION for current options.

4.1.2.1.5  Inputs, Outputs, and Other Data Elements

The inputs to mac_DDS are the parameters set in the DCS_CONFIGURATION table (see Table 9: DCS_CONFIGURATION). mac_DDS will have no outputs.

Function Check_for_schedule will have a directory listing as an input. It will have no outputs.

Function Check_disk_space will have stats from the local disk system, plus table DCS_RAWFILE_ACCT (see Table 10: DCS_RAWFILE_ACCT) and table TRANSFER_ACCT (see Table 11: TRANSFER_ACCT) for inputs. The function will have no outputs.

Function Check_for_xfers will have table DCS_RAWFILE_ACCT (see Table 10: DCS_RAWFILE_ACCT) and table TRANSFER_ACCT (see Table 11: TRANSFER_ACCT) for inputs. It will have no outputs.
4.1.2.1.6 Software Unit Logic

4.1.2.1.6.1 Initiation

mac_DDS will start up and launch the DDS GUI window. It will then go into a polling loop, executing the Check_for_schedule, Check_disk_space, and Check_for_xfers functions.

4.1.2.1.6.2 Control

mac_DDS will launch the DDS GUI window to run independently of mac_DDS (i.e. in the background).

Function Check_for_schedule will launch mac_IngestContact to ingest newly detected Contact Schedules. It will wait for mac_IngestContact to complete.

Function Check_disk_space will launch either mac_BackupArchive or mac_Delete if the disk space is below a threshold. It will launch mac_Delete for the oldest file on the system that has been received by all archiving systems. If disk space is low, and the remaining files have not been retrieved by all archiving systems, the newest file of the lowest priority (highest integer value) on the system is backed-up to tape via mac_BackupArchive.

4.1.2.1.6.3 Response

There are no special responses or response times to each input; including data conversion, renaming, and data transfer operations.

4.1.2.1.6.4 Sequence of Operations

mac_DDS:
1. Launch mac_ui_DDS_main to start the main GUI window.
2. Fork processes to begin polling loops (1 polling loop for each function):
   a. Poll/Execute function Check_for_schedule.
   b. Poll/Execute function Check_disk_space.
   c. Poll/Execute function Check_for_xfers.

Check_for_schedule:
1. Read options (polling loop interval) from DCS_CONFIGURATION table.
2. Launch ‘ls’ command to obtain list of files in schedule directory (which is "$DCS_HOME/schedules"), sorted to get oldest file first.
3. Loop through the files found,
   a. launch mac_IngestContact (which will load the information into the database, send updated support schedules to each CTS, and move the contact schedule file to a ‘processed’ directory).

Check_disk_space:
1. Read options (polling loop interval, disk threshold) from DCS_CONFIGURATION table.
2. Obtain available disk space for the raw file directory.
3. If the disk space is below the threshold
   a. Query database tables DCS_RAWFILE_ACCT, TRANSFER_ACCT, and
      DESTINATION_ACCT for a list of on-line raw files with lowest priority
      (highest integer value) that have been successfully archived (either
      transferred to all archiving systems defined for the mission, or copied onto
      a DCS Backup/Archive tape).
   b. if a file is found
      i. Launch mac_DeleteRawFiles to delete the newest file in the list
         (just one).

Check_for_xfers:
1. Read options (polling loop interval) from DCS_CONFIGURATION table.
2. Query database tables DCS_RAWFILE_ACCT and TRANSFER_ACCT for a
   list of on-line raw files successfully transferred to all destinations and the last
   TRANSFER_ACCT.STATUS_DATE is more than
   DCS_CONFIGURATION.DELETION_DELAY time old.
3. Launch mac_DeleteRawFiles to delete each file.
4. Select next available file for transfer (oldest on-line file of highest priority
   (lowest integer value)) for the destination and set the corresponding
   XFER_STATUS to ‘READY’.

4.1.2.1.6.4.1 Method of Sequence Control
The method for sequence control is defined by normal program flow within the
module.

4.1.2.1.6.4.2 Logic and Input Conditions
Each polling function will be executed as a separate process and will have it’s own
polling interval defined in the database.
Priority for automatic contact schedule ingest is to ingest the oldest (by file date)
first. Normally, only one contact schedule will appear in the schedule directory at
a time; however, if more than one should appear, the newest one should be
ingested last.
Priority for automatic clean-up is to remove files with lowest priority (highest integer
value) that are successfully archived. By removing files that are successfully
archived first, clean-up will be much faster. Also, it won’t make any difference to
processing systems whether they request the file from DCS Backup/Archive
tapes or from the archiving system (in fact requesting from the archiving systems
may be faster if the request is automated).

4.1.2.1.6.4.3 Data Transfer in and out Of Memory
The Data Transfer in and out Of Memory is not a concern
4.1.2.1.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.2.1.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.2.1.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.1.2.1.6.7 Details of the Software Unit

Figure 4-1 shows the basic execution flow for module mac_DDS.c

---

**Figure 4-1 mac_DDS Flowchart**
Figure 4-2 shows the basic execution flow for function Check_for_schedule.

Figure 4-2 Check_for_schedule Flowchart
Figure 4-3 shows the basic execution flow for function Check_disk_space.

read disk_poll and disk_threshold values from table DCS_CONFIG

check raw file disk available space

wait specified poll interval

log warning and wait 10 seconds

check database for on-line but archived files, sorted to get newest file of lowest priority.

polling interval > 0?

file found?

Launch mac_DeleteRawFile for newest file found

mac_DeleteRawFile

Log "low disk space" warning

Figure 4-3 Check_disk_space flowchart
Figure 4-4 shows the basic execution flow of function Check_for_xfers.

Figure 4-4 Check_for_xfers Flowchart
4.1.2.2 mac_IngestContact

4.1.2.2.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.1.2.2.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.2.2.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.1.2.2.4 Procedural Commands

mac_IngestContact will use Pro C embedded SQL to update the CONTACT_SCHEDULES table in the database.

4.1.2.2.5 Inputs, Outputs, and Other Data Elements

mac_IngestContact will read the contact schedule file given on the command line and update table CONTACT_SCHEDULES with the new information.

4.1.2.2.6 Software Unit Logic

4.1.2.2.6.1 Initiation

There are no special conditions in effect within the software unit when its execution is initiated.

4.1.2.2.6.2 Control

After ingesting the contact schedule, mac_IngestContact will launch mac_GenSuptSchedule to create and send CTS support schedules to the affected Capture Transfer Subsystems.

4.1.2.2.6.3 Response

There are no special responses or response times to each input; including data conversion, renaming, and data transfer operations.

4.1.2.2.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include the following:

4.1.2.2.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.
4.1.2.6.2 Logic and Input Conditions

mac_IngestContact will need to determine which mission is being updated by the contact schedule. For Landsat 7, field 1 of the Contact Schedule file is ‘7’, which easily determines the mission.

mac_IngestContact will need to look for passes defined in the old schedule with a non-default priority. It will then need to delete all mission contacts and insert new scheduled passes with default priority.

4.1.2.6.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern

4.1.2.6.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.2.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.2.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.1.2.6.7 Details of the Software Unit
Figure 4-5 shows the basic execution flow for module mac_IngestContact.

4.1.2.3 mac_GenSuptSchedule

4.1.2.3.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.1.2.3.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.2.3.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.
4.1.2.3.4 Procedural Commands

mac_GenSuptSchedule will use Pro C embedded SQL to query the database for schedule information.
mac_GenSuptSchedule will also use ftp to send the support schedule file to the CTS, and rsh to send a signal to the CTS.

4.1.2.3.5 Inputs, Outputs, and Other Data Elements

mac_GenSuptSchedule will receive a Mission id from the command line, then read from table CONTACT_SCHEDULE and create a support schedule file for each affected CTS.

4.1.2.3.6 Software Unit Logic

The software unit contains the following logic:

4.1.2.3.6.1 Initiation

There are no conditions in effect within the software unit when its execution is initiated.

4.1.2.3.6.2 Control

After a support schedule file is created, ftp will be launched to send the file to the CTS schedule directory. After the file is sent, mac_GenSuptSchedule will launch rsh to send a 'schedule update' signal to the CTS's rdc_AutoCapture process.

4.1.2.3.6.3 Response

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.1.2.3.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit's operation include:

4.1.2.3.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.1.2.3.6.4.2 Logic and Input Conditions

mac_GenSuptSchedule will need to determine which entries in table CONTACT_SCHEDULES to place in the support schedule. This is determined by the CAPTURE_MISSION_ACCT. Therefore, the CONTACT_SCHEDULES
table and CAPTURE_MISSION_ACCT table need to be setup prior to generating the support schedule.

4.1.2.3.6.4.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.

4.1.2.3.6.4.4 Discreet Inputs
There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.2.3.6.5 Exception and Error Handling
Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.2.3.6.6 Data Structure Charts
There are no significant data structures for this unit.

4.1.2.3.6.7 Details of the Software Unit
The general flow of mac_GenSuptSchedule is depicted in Figure 4-6.
4.1.2.4  mac_InsertFileNames

4.1.2.4.1  Unit Design Decisions

The support for IAS Calibration Parameter Files should be removed. Also, the temporary table used to hold the filenames should be renamed from LPS_CONTACT_SCHED_FILES to DCS_CONTACT_SCHED_FILES (to better reflect the architecture and reduce maintenance confusion).
4.1.2.4.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.2.4.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.1.2.4.4 Procedural Commands

Pro C embedded SQL is used to delete any existing entries and insert the filenames into the table.

4.1.2.4.5 Inputs, Outputs, and Other Data Elements

The module reads the schedule directory and inserts the filenames into the table.

4.1.2.4.6 Software Unit Logic

The software unit contains the following logic:

4.1.2.4.6.1 Initiation

There are no conditions in effect within the software unit when its execution is initiated.

4.1.2.4.6.2 Control

There are no conditions under which control is passed to other software units.

4.1.2.4.6.3 Response

There are no specific responses or response times for each input; including data conversion, renaming, and data transfer operations.

4.1.2.4.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.1.2.4.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.1.2.4.6.4.2 Logic and Input Conditions

mac_InsertFileNames will place all filenames in the schedules directory into table DCS_CONTACT_SCHED_FILES. Therefore, the files need to reside in the schedules directory prior to executing mac_InsertFileNames.
4.1.2.4.6.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.

4.1.2.4.6.4 Discreet Inputs
There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.2.4.6.5 Exception and Error Handling
Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.2.4.6.6 Data Structure Charts
There are no significant data structures for this unit.

4.1.2.4.6.7 Details of the Software Unit
The detailed design of this software unit is in the existing code (no changes required).

4.1.2.5 mac_SendSetup

4.1.2.5.1 Unit Design Decisions
There are no special design decisions, such as algorithms used, for this unit.

4.1.2.5.2 Constraints, Limitations, or Unusual Features
There are no special constraints, limitations, or unusual features for this unit.

4.1.2.5.3 Programming Language
This unit will be implemented using the ANSI standard C programming language.

4.1.2.5.4 Procedural Commands
Pro C embedded SQL is used to access database records.

4.1.2.5.5 Inputs, Outputs, and Other Data Elements
The module reads the database records and inserts the CTS setup information into an ASCII file to be sent to the CTS.

4.1.2.5.6 Software Unit Logic
The software unit contains the following logic:
4.1.2.5.6.1 Initiation
There are no conditions in effect within the software unit when its execution is initiated.

4.1.2.5.6.2 Control
Control will be passed to the UNIX ftp command to send the setup file, and then to the UNIX rsh command to signal the CTS’s rdc_AutoCapture process to re-read the setup information.

4.1.2.5.6.3 Response
There are no specific responses or response times for each input; including data conversion, renaming, and data transfer operations.

4.1.2.5.6.4 Sequence of Operations
The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.1.2.5.6.4.1 Method of Sequence Control
The method for sequence control is defined by normal program flow within the module.

4.1.2.5.6.4.2 Logic and Input Conditions
mac_SendSetup will create a temporary ASCII setup file for a CTS, therefore the setup information records need to exist in the database.

4.1.2.5.6.4.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.

4.1.2.5.6.4.4 Discreet Inputs
There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.2.5.6.5 Exception and Error Handling
Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.2.5.6.6 Data Structure Charts
There are no significant data structures for this unit.
4.1.2.5.6.7 Details of the Software Unit

1. Read command-line parameters and validate (does specified CTS exist in database?).
2. Read parameters from DCS_CAPTURE_ACCT.
3. Read channel mapping from DCS_CAPT_CHAN_MAP_ACCT.
4. Write parameters to ASCII setup file.
5. Read parameters from MISSION_ACCT.
6. Write parameters to ASCII mission info file.
7. ftp login to specified CTS.
8. Send files to CTS.
10. rsh ‘kill’ command to send signal SIGHUP to rdc_AutoCapture on CTS.

4.1.2.6 mac_ManCapture

4.1.2.6.1 Unit Design Decisions
There are no special design decisions, such as algorithms used, for this unit.

4.1.2.6.2 Constraints, Limitations, or Unusual Features
There are no special constraints, limitations, or unusual features for this unit.

4.1.2.6.3 Programming Language
This unit will be implemented using the ANSI standard C programming language.

4.1.2.6.4 Procedural Commands
Pro C embedded SQL is used to access database records.

4.1.2.6.5 Inputs, Outputs, and Other Data Elements
The module reads the command-line parameters and inserts the capture information into an ASCII file to be sent to the CTS.

4.1.2.6.6 Software Unit Logic
The software unit contains the following logic:

4.1.2.6.6.1 Initiation
There are no conditions in effect within the software unit when its execution is initiated.
4.1.2.6.2 Control

Control will be passed to the UNIX ftp command to send the capture parameter file, and then to the UNIX rsh command to signal the CTS’s rdc_AutoCapture process to read the capture information and perform the capture.

4.1.2.6.3 Response

There are no specific responses or response times for each input; including data conversion, renaming, and data transfer operations.

4.1.2.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.1.2.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.1.2.6.4.2 Logic and Input Conditions

mac_ManCapture will create a temporary ASCII setup file for a CTS from command-line parameters, therefore the command-line parameters will need to be specified and validated, or suitable defaults used.

4.1.2.6.4.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.

4.1.2.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.2.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.2.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.1.2.6.7 Details of the Software Unit

1. Read command-line parameters and validate (does specified CTS exist in database?).
2. Write parameters to ASCII capture file.
3. ftp login to specified CTS.
4. Send file to CTS.
5. Log-out.
6. rsh kill command to signal rdc_AutoCapture on CTS.

4.1.3 Back-up Archive Component

4.1.3.1 mac_BackupArchive

4.1.3.1.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.1.3.1.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.3.1.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.1.3.1.4 Procedural Commands

This module will use Pro C embedded SQL to update database records.

4.1.3.1.5 Inputs, Outputs, and Other Data Elements

The module will update table BACKUP_ACCT.

4.1.3.1.6 Software Unit Logic

The software unit contains the following logic:

4.1.3.1.6.1 Initiation

Conditions that may be in affect when mac_BackupArchive is initiated:
1. tape is being reused (i.e. at BOT, with registered tape-id)
2. tape is new (i.e. at BOT, no tape-id header).
3. tape has unregistered data (i.e. at BOT, with unregistered tape-id or tape header date is newer than registered id)
4. tape is ready for appending another file (i.e. not at BOT, with room for file).
5. tape is full (i.e. not at BOT, no room left for file).
6. no tape in drive.

4.1.3.1.6.2 Control

mac_BackupArchive will use rdc_Save to perform the copy to tape.
4.1.3.1.6.3 **Response**

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.1.3.1.6.4 **Sequence of Operations**

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.1.3.1.6.4.1 **Method of Sequence Control**

The method for sequence control is defined by normal program flow within the module.

4.1.3.1.6.4.2 **Logic and Input Conditions**

mac_BackupArchive will copy the indicated file to the backup tape device. Therefore, the file must exist and the tape device must be ready.

4.1.3.1.6.4.3 **Data Transfer in and out Of Memory**

The data transfer in and out for memory is not a concern.

4.1.3.1.6.4.4 **Discreet Inputs**

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.3.1.6.5 **Exception and Error Handling**

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.3.1.6.6 **Data Structure Charts**

There are no significant data structures for this unit.

4.1.3.1.6.7 **Details of the Software Unit**

The basic flow of execution of the mac_BackupArchive software unit is depicted in Figure 4-7.
4.1.4 Delete Raw Files Component (DDS)

4.1.4.1 mac_DeleteRawFiles

4.1.4.1.1 Unit Design Decisions
There are no special design decisions, such as algorithms used, for this unit.

4.1.4.1.2 Constraints, Limitations, or Unusual Features
There are no special constraints, limitations, or unusual features for this unit.

4.1.4.1.3 Programming Language
This unit will be implemented using the ANSI standard C programming language.

4.1.4.1.4 Procedural Commands
mac_DeleteRawFiles will use Pro C embedded SQL to update the
DCS_RAWFILE_ACCT table to show that the file is no longer on-line.

4.1.4.1.5 Inputs, Outputs, and Other Data Elements
The module updates table DCS_RAWFILE_ACCT to set the ON_LINE_FLAG.

4.1.4.1.6 Software Unit Logic
The software unit contains the following logic:

4.1.4.1.6.1 Initiation
Special conditions that may be in affect when mac_DeleteRawFiles is initiated:

1. The file may not be eligible for deletion. Eligibility is met by one of the following:
a. The file is delivered to all archiving destinations.
b. The file is backed up to tape (in the DCS Backup/Archive).

2. The option to override eligibility checking was given.

4.1.4.1.6.2 Control
There are no transfers of control to other processing modules.

4.1.4.1.6.3 Response
There are no specific responses or response times for each input, including data
corversion, renaming, and data transfer operations.

4.1.4.1.6.4 Sequence of Operations
The sequence of operations and dynamically controlled sequencing during the
software unit's operation include:

4.1.4.1.6.4.1 Method of Sequence Control
The method for sequence control is defined by normal program flow within the
module.

4.1.4.1.6.4.2 Logic and Input Conditions
mac_DeleteRawFiles will delete the raw file from on-line storage. Since the file
may not be archived, mac_DeleteRawFiles will verify with the database that the
file is archived prior to deletion.

4.1.4.1.6.4.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.

4.1.4.1.6.4.4 Discreet Inputs
There is no sensing of discreet input signals, and timing relationships between
interrupt operations within this software unit.

4.1.4.1.6.5 Exception and Error Handling
Exception and Error Handling will normally log appropriate journal messages, but
continue with normal operations (if possible).

4.1.4.1.6.6 Data Structure Charts
There are no significant data structures for this unit.
4.1.4.1.6.7 Details of the Software Unit

Figure 4-8 mac_DeleteRawFiles Flowchart

4.1.5 Restage from Tape Component

4.1.5.1 mac_Restage

4.1.5.1.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.1.5.1.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.5.1.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.
4.1.5.1.4 Procedural Commands

This module will use Pro C embedded SQL to update database records.

4.1.5.1.5 Inputs, Outputs, and Other Data Elements

The module does not contain, receive, or output data.

4.1.5.1.6 Software Unit Logic

The software unit contains the following logic:

4.1.5.1.6.1 Initiation

Conditions that may be in affect when mac_Restage is initiated:
1. tape has registered data (i.e. with registered tape-id and header date is same as registered header date)
2. tape has unregistered data (i.e. with unregistered tape-id or tape header date is newer than registered id)
3. tape not rewound.
4. no tape in drive (or tape without header).

4.1.5.1.6.2 Control

mac_Restage will use tar to perform the copy from tape.

4.1.5.1.6.3 Response

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.1.5.1.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.1.5.1.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.1.5.1.6.4.2 Logic and Input Conditions

mac_Restage will either copy the selected raw file from tape, or all raw files. Therefore, the tape device must be ready and the local disk must have sufficient available storage to hold the file.

4.1.5.1.6.4.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.
4.1.5.1.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between
interrupt operations within this software unit.

4.1.5.1.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but
continue with normal operations (if possible).

4.1.5.1.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.1.5.1.6.7 Details of the Software Unit

Figure 4-9 depicts the top-level execution flow for module rdc_Restage.

NOTE: restage will place the rawfile into the DDS incoming raw file
directory. mac_UpdDCSAcct will move the rawfile to the mission-
specific rawfile directory.
Figure 4-9 mac_Restage Flowchart
4.1.5.2  mac_UpdDCSAcct

4.1.5.2.1  Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.1.5.2.2  Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.5.2.3  Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.1.5.2.4  Procedural Commands

This module will use Pro C embedded SQL to update database records.

4.1.5.2.5  Inputs, Outputs, and Other Data Elements

The module creates/updates records in table DCS_RAWFILE_ACCT and table TRANSFER_ACCT.

4.1.5.2.6  Software Unit Logic

The software unit contains the following logic:

4.1.5.2.6.1  Initiation

Conditions that may be in affect when mac_UpdDCSAcct is initiated:
1. The DDS incoming rawfile directory may contain unregistered data. The following will need to be performed:
   a. The raw file will need to be moved to the mission-specific rawfile directory.
   b. The accounting files will need to be ingested into the database
   c. The corresponding TRANSFER_ACCT records will need to be created.
2. The raw data capture information may conflict with a previous raw data file (e.g. the Scheduled_Start_Time, Original_Station, and Capture_Source should uniquely identify the raw file).

4.1.5.2.6.2  Control

mac_UpdDCSAcct will pass control to another process.

4.1.5.2.6.3  Response

There are no specific responses or response times for each input; including data conversion, renaming, and data transfer operations.
4.1.5.2.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.1.5.2.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.1.5.2.6.4.2 Logic and Input Conditions

mac_UpdDCSAcct will create database records (in DCS_RAWFILE_ACCT and TRANSFER_ACCT) for new raw files.

4.1.5.2.6.4.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.

4.1.5.2.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.5.2.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.5.2.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.1.5.2.6.7 Details of the Software Unit

Figure 4-10 presents the top-level execution of module mac_UpdDCSAcct.
Figure 4-10 mac_UpdDCSAcct Flowchart
4.1.6   Journaling Component

4.1.6.1   mac_JournalFileEntry

4.1.6.1.1   Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.1.6.1.2   Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.1.6.1.3   Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.1.6.1.4   Procedural Commands

There are no special procedural commands (such as menu selections in a 
database management system (DBMS) for defining forms and exports, on-line 
DBMS queries for database access and manipulation, input to a graphical user 
interface (GUI) builder for automated code generation, commands to the 
operating system, or shell scripts).

4.1.6.1.5   Inputs, Outputs, and Other Data Elements

The module will log the given message to the journal.

4.1.6.1.6   Software Unit Logic

The software unit contains the following logic:

4.1.6.1.6.1   Initiation

There are no special conditions in effect within the software unit when its execution 
is initiated.

4.1.6.1.6.2   Control

There are no conditions under which control is passed to other software units.

4.1.6.1.6.3   Response

There are no specific responses or response times for each input; including data 
conversion, renaming, and data transfer operations.

4.1.6.1.6.4   Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the 
software unit’s operation include:
4.1.6.1.6.1 Method of Sequence Control
The method for sequence control is defined by normal program flow within the module.

4.1.6.1.6.2 Logic and Input Conditions
The journaling component will utilize the Unix syslog facility. This facility will need to be properly setup by the system administrator in order for the messages to be logged.

4.1.6.1.6.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.

4.1.6.1.6.4 Discreet Inputs
There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.1.6.1.6.5 Exception and Error Handling
Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.1.6.1.6.6 Data Structure Charts
There are no significant data structures for this unit.

4.1.6.1.6.7 Details of the Software Unit
The mac_JournalFileEntry process will be reused without modification.

4.2 Capture Transfer Subsystem

4.2.1 AutoCapture Component

4.2.1.1 rdc_AutoCapture

4.2.1.1.1 Unit Design Decisions
There are no special design decisions, such as algorithms used, for this unit.

4.2.1.1.2 Constraints, Limitations, or Unusual Features
There are no special constraints, limitations, or unusual features for this unit.
4.2.1.3 Programming Language

This unit is implemented using the ANSI standard C programming language.

4.2.1.4 Procedural Commands

There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.1.5 Inputs, Outputs, and Other Data Elements

The module does not contain, receive, or output data.

4.2.1.6 Software Unit Logic

The software unit contains the following logic:

4.2.1.6.1 Initiation

There are no special conditions in effect within the software unit when its execution is initiated.

4.2.1.6.2 Control

There are no conditions under which control is passed to other software units.

4.2.1.6.3 Response

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.2.1.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit's operation include:

4.2.1.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.2.1.6.4.2 Logic and Input Conditions

There are no special logic and input conditions of the method, such as timing variations or priority assignments.
4.2.1.6.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.

4.2.1.6.4 Discreet Inputs
There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.1.6.5 Exception and Error Handling
Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.1.6.6 Data Structure Charts
There are no significant data structures for this unit.

4.2.1.6.7 Details of the Software Unit
Figure 4-11 shows the top-level execution flow for module rdc_AutoCapture.
4.2.1.2 rdc_AutoTransfer

4.2.1.2.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.
4.2.1.2.2 **Constraints, Limitations, or Unusual Features**

There are no special constraints, limitations, or unusual features for this unit.

4.2.1.2.3 **Programming Language**

This unit is implemented using the ANSI standard C programming language.

4.2.1.2.4 **Procedural Commands**

There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.1.2.5 **Inputs, Outputs, and Other Data Elements**

The module does not contain, receive, or output data.

4.2.1.2.6 **Software Unit Logic**

The software unit contains the following logic:

4.2.1.2.6.1 **Initiation**

There are no special conditions in effect within the software unit when its execution is initiated.

4.2.1.2.6.2 **Control**

There are no conditions under which control is passed to other software units.

4.2.1.2.6.3 **Response**

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.2.1.2.6.4 **Sequence of Operations**

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.2.1.2.6.4.1 **Method of Sequence Control**

The method for sequence control is defined by normal program flow within the module.
4.2.1.2.6.4.2 Logic and Input Conditions

There are no special logic and input conditions of the method, such as timing variations or priority assignments.

4.2.1.2.6.4.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.

4.2.1.2.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.1.2.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.1.2.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.2.1.2.6.7 Details of the Software Unit

Figure 4-11 shows the top-level execution flow for module rdc_AutoTransfer.
Figure 4-12 rdc_AutoTransfer Flowchart
4.2.2 Raw Data Capture Component

4.2.2.1 rdc_Capture

4.2.2.1.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.2.2.1.2 Constraints, Limitations, or Unusual Features

rdc_Capture makes use of some IRIX-specific realtime features (see the REACT man page). Some of these features will require special privileges.

4.2.2.1.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.2.2.1.4 Procedural Commands

There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.2.1.5 Inputs, Outputs, and Other Data Elements

rdc_Capture reads one stream of raw wideband data received via the Myriad Logic capture device and writes the raw data directly to a disk file. The module also creates an associated accounting file.

4.2.2.1.6 Software Unit Logic

The software unit contains the following logic:

4.2.2.1.6.1 Initiation

The Myriad Logic serial capture is expected to be ready to receive the wideband data. All non-essential processes that may interfere with capture are expected to be suspended.

4.2.2.1.6.2 Control

There are no conditions under which control is passed to other software units.

4.2.2.1.6.3 Response

rdc_Capture needs to read and write raw data at a minimum rate of 75 Megabits per second (approximately 9 megabytes per second).
4.2.2.1.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.2.2.1.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.2.2.1.6.4.2 Logic and Input Conditions

rdc_Capture will write a stream of incoming wideband data to disk. The disk needs to have sufficient available storage for the incoming data.

4.2.2.1.6.4.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.

4.2.2.1.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.2.1.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.2.1.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.2.2.1.6.7 Details of the Software Unit

Figure 4-13 shows the top-level execution flow for module rdc_Capture.
Figure 4-13 rdc_Capture Flowchart

4.2.3 Transfer To Tape Component

4.2.3.1 rdc_Save

4.2.3.1.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.2.3.1.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.
4.2.3.1.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.2.3.1.4 Procedural Commands

There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.3.1.5 Inputs, Outputs, and Other Data Elements

The module does not contain, receive, or output data.

4.2.3.1.6 Software Unit Logic

The software unit contains the following logic:

4.2.3.1.6.1 Initiation

Conditions that may be in affect when rdc_Save is initiated:

1. tape is being reused (i.e. at BOT, with registered tape-id)
2. tape is new (i.e. at BOT, no tape-id header)
3. tape has unregistered data (i.e. at BOT, with unregistered tape-id or tape header date is newer than registered id)
4. tape is ready for appending another file (i.e. not at BOT, with room for file)
5. tape is full (i.e. not at BOT, no room left for file)
6. no tape in drive

4.2.3.1.6.2 Control

The software unit utilizes the ‘tar’ command to write the files to tape.

4.2.3.1.6.3 Response

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.2.3.1.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.2.3.1.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.
4.2.3.1.6.4.2 Logic and Input Conditions

rdc_Save will copy the indicated file to tape. Therefore, the tape device must be ready to received the file.

4.2.3.1.6.4.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.

4.2.3.1.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.3.1.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.3.1.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.2.3.1.6.7 Details of the Software Unit
4.2.3.2 rdc_GenStackedLabel

4.2.3.2.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.2.3.2.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.2.3.2.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.2.3.2.4 Procedural Commands

There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line
DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.3.2.5 Inputs, Outputs, and Other Data Elements

The module does not contain, receive, or output data.

4.2.3.2.6 Software Unit Logic

The software unit contains the following logic:

4.2.3.2.6.1 Initiation

Conditions that may be in affect when rdc_Save is initiated:
1. tape is being reused (i.e. at BOT, with registered tape-id)
2. tape is new (i.e. at BOT, no tape-id header).
3. tape has unregistered data (i.e. at BOT, with unregistered tape-id or tape header date is newer than registered id)
4. tape is ready for appending another file (i.e. not at BOT, with room for file).
5. tape is full (i.e. not at BOT, no room left for file).
6. no tape in drive.

4.2.3.2.6.2 Control

The software unit utilizes the ‘tar’ command to write the files to tape.

4.2.3.2.6.3 Response

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.2.3.2.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.2.3.2.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.2.3.2.6.4.2 Logic and Input Conditions

rdc_Save will copy the indicated file to tape. Therefore, the tape device must be ready to received the file.
4.2.3.2.6.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.

4.2.3.2.6.4 Discreet Inputs
There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.3.2.6.5 Exception and Error Handling
Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.3.2.6.6 Data Structure Charts
There are no significant data structures for this unit.

4.2.3.2.6.7 Details of the Software Unit
Summary of New Module rdc_GenStackedLabel.c:
1) If tape id and hostname are given on the command line
   a) Fetch id # from command line.
   b) Fetch hostname from command line.
2) else
   a) Fetch id # from tape_id file, and increment.
   b) Fetch hostname from system.
3) Print tape label (Project Name, hostname-tape_id, date)

4.2.4 Delete Raw Files Component (CTS)
4.2.4.1 rdc_Delete

4.2.4.1.1 Unit Design Decisions
There are no special design decisions, such as algorithms used, for this unit.

4.2.4.1.2 Constraints, Limitations, or Unusual Features
There are no special constraints, limitations, or unusual features for this unit.

4.2.4.1.3 Programming Language
This unit will be implemented using the ANSI standard C programming language.

4.2.4.1.4 Procedural Commands
There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line
DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.4.1.5 Inputs, Outputs, and Other Data Elements
The module does not contain, receive, or output data.

4.2.4.1.6 Software Unit Logic
The software unit contains the following logic:

4.2.4.1.6.1 Initiation
The file to be deleted may have read-only protection, which would indicate that the file has not been sent to DDS or copied to tape. If the file does not have read-only protection, the presumption is that it has been either sent to DDS or copied to tape.

4.2.4.1.6.2 Control
There are no conditions under which control is passed to other software units.

4.2.4.1.6.3 Response
There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.2.4.1.6.4 Sequence of Operations
The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:

4.2.4.1.6.4.1 Method of Sequence Control
The method for sequence control is defined by normal program flow within the module.

4.2.4.1.6.4.2 Logic and Input Conditions
rdc_Delete will delete the raw data file and its associated accounting file. Therefore, the permissions must be set on the files to allow the deletion.

4.2.4.1.6.4.3 Data Transfer in and out Of Memory
The data transfer in and out for memory is not a concern.
4.2.4.1.6.4 Discreet Inputs

There are no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.4.1.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.4.1.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.2.4.1.6.7 Details of the Software Unit

![Flow Chart]

Figure 4-15 rdc_DeleteFiles Flow Chart

4.2.5 Raw File Transfer Component (CTS)

4.2.5.1 rdc_Transfer

4.2.5.1.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.
4.2.5.1.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.2.5.1.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.2.5.1.4 Procedural Commands

There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.5.1.5 Inputs, Outputs, and Other Data Elements

The module does not contain, receive, or output data.

4.2.5.1.6 Software Unit Logic

The software unit contains the following logic:

4.2.5.1.6.1 Initiation

There are no special conditions in effect within the software unit when its execution is initiated.

4.2.5.1.6.2 Control

The software unit utilizes the 'mac_UpdDCSAcc' command to notify the DDS of available files.

4.2.5.1.6.3 Response

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.2.5.1.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit's operation include:

4.2.5.1.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.
4.2.5.1.6.2 Logic and Input Conditions

rdc_Transfer will send the raw data file and its associated accounting file to the DDS, as well as executing the mac_UpdDCSAcct to register the new file on the DDS. Therefore, the DDS must be ready to receive the file, mac_UpdDCSAcct must be executable, and the DCS database must be ready.

4.2.5.1.6.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.

4.2.5.1.6.4 Discreet Inputs

There are no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.5.1.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.5.1.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.2.5.1.6.7 Details of the Software Unit

![Figure 4-16 rdc_TransferFile Flow Chart](image-url)

9/10/09

- 129 -

DCS 104.4
4.2.6 Raw Data Transmit Component

4.2.6.1 rdc_Transmit

4.2.6.1.1 Unit Design Decisions

There are no special design decisions, such as algorithms used, for this unit.

4.2.6.1.2 Constraints, Limitations, or Unusual Features

There are no special constraints, limitations, or unusual features for this unit.

4.2.6.1.3 Programming Language

This unit will be implemented using the ANSI standard C programming language.

4.2.6.1.4 Procedural Commands

There are no special procedural commands (such as menu selections in a database management system (DBMS) for defining forms and exports, on-line DBMS queries for database access and manipulation, input to a graphical user interface (GUI) builder for automated code generation, commands to the operating system, or shell scripts).

4.2.6.1.5 Inputs, Outputs, and Other Data Elements

The module does not contain, receive, or output data.

4.2.6.1.6 Software Unit Logic

The software unit contains the following logic:

4.2.6.1.6.1 Initiation

There are no special conditions in effect within the software unit when its execution is initiated.

4.2.6.1.6.2 Control

There are no conditions under which control is passed to other software units.

4.2.6.1.6.3 Response

There are no specific responses or response times for each input, including data conversion, renaming, and data transfer operations.

4.2.6.1.6.4 Sequence of Operations

The sequence of operations and dynamically controlled sequencing during the software unit’s operation include:
4.2.6.1.6.4.1 Method of Sequence Control

The method for sequence control is defined by normal program flow within the module.

4.2.6.1.6.4.2 Logic and Input Conditions

rdc_Transmit will copy the indicated file out through the Myriad Logic high-speed serial device. Therefore, the file must be readable and the serial device must be ready for transmit.

4.2.6.1.6.4.3 Data Transfer in and out Of Memory

The data transfer in and out for memory is not a concern.

4.2.6.1.6.4.4 Discreet Inputs

There is no sensing of discreet input signals, and timing relationships between interrupt operations within this software unit.

4.2.6.1.6.5 Exception and Error Handling

Exception and Error Handling will normally log appropriate journal messages, but continue with normal operations (if possible).

4.2.6.1.6.6 Data Structure Charts

There are no significant data structures for this unit.

4.2.6.1.6.7 Details of the Software Unit

There are no changes planned for this software unit (other than updating the rdc_DeviceFunctions module for the Myriad Logic capture card).

4.2.7 Journaling Component

The Journaling Component on the CTS will reuse the journaling component on the DDS.

4.3 MWD Formatter Component

4.3.1 Purpose

The MWD formatter component is responsible for accepting one or more serial streams of raw data from the RDC component and producing formatted output for the MWD component. The current design of the MWD formatter component will support two data formats:
• Enhanced Thematic Mapper Plus (ETM+) data from Landsat 7 (L7)
• Thematic Mapper (TM) from Landsat 5 (L5).

For convenience, these will be called L7 format and L5 format in the rest of this discussion. These data streams are generally acquired in real-time from a ground station and forwarded to the CTS for processing. However, the data streams may also be played back at a later time from a recorder in the receiving equipment. The design of this system will handle both data acquired in real time, and data that is played back later.

The MWD Formatter component is an application that is initially run as a child process of the rdc_AutoCapture component. When started, the application is passed the name of an initialization file that configures the application to process format that will be acquired. The MWD Formatter will then start up one or more processing “strings” that handle a data stream. For the L7 format there are two streams, one for the I channel, and the other for the Q channel. For the L5 formatter, there is a single data stream.

Each of the processing “strings” will be made up of 3 processing objects: an ingest object, a frame sync object, and a formatter object. The ingest object will acquire its data from a ring buffer that is represented by an “RDC” object. Between the ingest and frame sync objects there will be a FIFO object. Similarly, between the frame sync and formatter objects there will also be a FIFO object. These FIFO objects are synchronized ring buffers that are used to temporarily stack data as needed. The formatter object will write its data to an MWD object. Figure X.1 depicts this architecture.

Each of the three processing objects may construct multiple internal objects including multiple thread objects. Minimally, there will be one distinct thread for each of the three main objects. Each of the objects may instantiate additional threads as necessary. The reason for running these multiple threads is to spread the workload out on a multiprocessor machine. Both the frame synchronization and the formatting functions are CPU intensive. By running these two on separate threads they will run on separate processors (as needed) to provide more overall throughput.
4.3.2 Subcomponent Design

As already implied, object-oriented programming will be used to implement the MWD formatter. The 7 main subcomponents (Control, Ingest, Frame Sync, Formatter, RDC, FIFO, and MWD) will all be implemented as objects. For the Frame Sync and Formatter objects, there is a different subclass for each data format. There are numerous other supporting objects that will also be used in the implementation of this system. Each of the following subsections will describe the functionality provided by a single source file of the system. In many cases, this source will consist of a single object, but in others there will be a set of related objects.

4.3.2.1 MWDFormatter

This source will contain the ‘main’ method. For this system, the main method is very simple. It will:

1. Parse the command line to get the initialization file name and the CTD to use.
2. Initialize the logging system.
3. Call the static “initialize” method of the Control class.
4. Call static “initialize” methods for each of the main subcomponents.

*Figure 4-17 Processing String*
5. Call static "run" method for the control class
6. Exit

The main method will look something like this (some code is excluded for clarity):

```cpp
int main(int argc, char* argv[]) {
    // parse command line
    ..
    // initialize logging
    ..
    // Initialize the control system
    Control::initialize(initfile);
    .. // Initialize ingest subcomponents
    RDCIngest::initialize(initfile); // For Operations
    DiskIngest::initialize(initfile); // For Testing
    .. // initialize frame sync subcomponents
    L7FrameSync::initialize(initfile);
    L5FrameSync::initialize(initfile);
    .. // initialize formatter subcomponents
    L7Formatter::initialize(initfile);
    L5Formatter::initialize(initfile);
    .. // start processing
    ..int status = Control::run();
    return status;
}
```

The command line syntax for running this application will be:

```
MWDFormatter [-c <initfile>] [-d <display>] [-f <streamfile>]
```

- **initfile** is the name of the initialization file to use. It is expected that there will be at least 4 initialization files one set for RDCIngest, one for DiskIngest, each with a L7 and an L7 variant.
- **display** is the number of the CTD to run. Used in configuration with configuration information in the initfile to determine the IP address to use for the display PC
- **streamfile** is a pathname to a bitfile, used for testing purposes only. Will only be used if a DiskIngest is being run.

The Control **initialize** method will create the FIFO and MWD components needed for processing the format.

Each of the other significant subcomponents will parse the initialization file to find out how many instances of the object to create. For each instance to be created, the object is created using initialization information for the object from the initialization file. Each object is then "plugged in" to the Modules class so that it can be accessed by the Control or other objects.
The design for this release of the MWDFormatter is limited to processing two data formats, L7 and L5. However, there is an implied requirement for the CTS to be expandable to other data formats in the future. To add other formats, all that will be required is to write new descendent classes of the abstract Frame Sync and Formatter classes to implement the code specific to those formats, and then to call the static “initialize” methods of these classes in the main function. In other words, the only existing source that will need to be modified is the MWDFormatter source.

### 4.3.2.2 Main Classes

#### 4.3.2.2.1 Modules

The Modules class acts as a global area for the main objects of the system so that they can see each other. This class will be comprised entirely of static methods. Internally, there will be 6 vectors (see the section describing Containers class) into which the main modules of the system will be inserted. These are:

- Ingest modules
- Raw data FIFOs
- Frame Sync modules
- Frame Data FIFOs
- Formatter Modules
- MWD Interfaces

Each of these vectors will hold a single entry for L5 data, and two entries for L7 data. The FIFO and MWD Interface objects will be constructed by the Control class based on the contents of the initialization file. After construction these objects will be inserted into the Modules arrays using one of the addXXX methods. To retrieve a reference to one of the objects, a call to the getXXX method is used. The following is a list of all the methods provided:

```c
// Add methods
// return instance number 1-N
static int addIngest(Ingest* ingest);
static int addRawFIFO(FIFO* fifo);
static int addFrameSync(FrameSync* framesync);
static int addFrameFIFO(FIFO* fifo);
static int addFormatter(Formatter* formatter);

// Get methods
// accept instance number, return object reference
static Ingest* getIngest(int number);
static FIFO* getRawFIFO(int number);
static FrameSync* getFrameSync(int number);
static FIFO* getFrameFIFO(int number);
static Formatter* getFormatter(int number);
```

The primary use of the Modules class is for the Control class to find the objects of interest. Note that the Ingest, FrameSync, and Formatter object references in the Modules class are of abstract class types, so the Control and other classes referencing
these objects do not know the actual type. This allows any type of Ingest, FrameSync, or Formatter module to be plugged into the infrastructure in support of new formats in the future.

One other notable use of the Modules class is for channel swaps in the L7 data format. It is never known beforehand which of the two serial streams (I or Q) for L7 data contains which channel (1 or 2). It is only after the L7FrameSync extracts the frames from the raw data that the channel number is known. The L7FrameSync will place the frame data into the FrameFIFO with the instance number corresponding to the channel number. It does this by doing something like:

```c++
// Get the correct output FIFO to use for the channel
FIFO* fifo = Modules::getFrameFIFO(channel_number);
// Write the frame to the FIFO
int fifo->write(frame);
```

4.3.2.2 Control

The Control class is responsible for coordinating the activities of the three main objects that are required to produce MWD output from a serial stream. For the L7 format, the Control object will be responsible for managing two sets of the three objects, one for each data stream.

The static `initialize` method of the control class will first read the initialization file to determine how many FIFO and MWD objects are to be constructed. Each constructed object will be placed into a FIFO or an MWD object array in the Modules class. The initialization method must be called before all other main class `initialize` methods.

After all main class `initialize` methods are called, the static `run` method of the control class will be called. The `run` method performs the following steps:

1. It calls the `prepare` methods for each object in forward order (Ingest, Frame Sync, then Formatter). For L7, the methods are called in pairs, both Ingests, then both Frame Syncs, then both Formatters. These methods open files, start threads, and do other initialization required for the formatting operation. If any error is returned from these methods, the formatting operation is shutdown.
2. It calls the `go` methods for each object in reverse order (Formatter, Frame Sync, then Ingest). Again, these methods are called in pairs for L7 processing. These methods start the processing. No errors are expected to be returned here.
3. It loops, repeatedly calling the `isStopped` method for each of the objects to see if the processing has completed. This loop will independently check for stopping on the two sides for L7. For L7, the loop will continue until at least one object of both sets has indicated it has stopped.
4. It calls the `stop` method for each object. These methods clean up processing, closing any files that are open.

The `run` method will log any errors that it finds during processing to the log file. Again, for L7, processing will continue until both streams have stopped or failed. If any error is found during processing, the `run` method will return a negative value. Otherwise, the `run` method returns 0 for success.

A synopsis of the two entry points for the Control class follows:

```c
// public methods defined by control
static int initialize(InitFile *initfile);
static int run();
```

### 4.3.2.2.3 FIFO

FIFOs are used to pass fixed length blocks of data between shared processes created using the Thread class (described in a later section). These routines utilize shared data areas for passing the data and standard Thread synchronization methods, so they will not work for processes created without using the Thread class. The amount of data used by the FIFO is the number of blocks in the FIFO times the size of each block. There are no coded restrictions on size, but obviously there are practical limits imposed by the computer hardware and operating system.

A FIFO is a rotating buffer of data blocks that is filled by one or more writer processes and read by one or more reader processes. Writers and readers are both referred to as users.

Writers are allowed to pass the block number they are writing to the FIFO routines. This way, multiple writers can use the FIFO at the same time. A writer attempting to write a block other than the next block that goes into the FIFO will wait until the other writers supply the intervening blocks. A write will also have to wait for space to become available in a FIFO in the event all blocks are filled. For single writer applications, writers can use the form of the write call that does not have a block number to write the next block.

There are two types of readers (as defined by the FIFO attach call). The first type (AttachReadAll) is a process that only wants to read every block from from the FIFO. The second (AttachReadNext) is more complex, and that is a process that intends to interleave access with all other AttachReadNext readers of the FIFO. It is possible to have both types attached to the same FIFO. However, it is not possible to have multiple sets of AttachReadNext readers that collectively get all blocks (in other words, there is exactly one set of AttachReadNext readers that collectively get all blocks). Readers have the option to wait or not wait when there is no data available.
The following table presents a synopsis of the FIFO methods that are publicly available:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO(int blocksize, int blockcount)</td>
<td>Construct the FIFO</td>
</tr>
<tr>
<td>int attach(FIFOAttachType attachtype)</td>
<td>Attach to a FIFO, attachtype is one of: AttachWrite, Attach for writing, AttachReadAll, Attach to read all blocks, AttachReadNextAttach to read next block</td>
</tr>
<tr>
<td>int readBlock(void* block, bool nowait)</td>
<td>Read a block from the FIFO, wait if no blocks are available and nowait is true, returns block number or –1 if no blocks available</td>
</tr>
<tr>
<td>int writeBlock(void* block, int blocknumber)</td>
<td>Write specified block number to FIFO, waits if FIFO is full</td>
</tr>
<tr>
<td>int write(void* block)</td>
<td>Write next block to FIFO</td>
</tr>
<tr>
<td>int getBlockSize(void) const</td>
<td>Get the size of a block in the FIFO</td>
</tr>
<tr>
<td>int getBlockCount(void) const</td>
<td>Get the number of blocks in the FIFO</td>
</tr>
<tr>
<td>void shutdown(void)</td>
<td>Shutdown the FIFO, causes all FIFO uses to get a FIFO_STATUS_SHUTDOWN on next read/write attempt</td>
</tr>
<tr>
<td>~FIFO(void)</td>
<td>Destructor, cleans up, should only be used when no more read/write attempts will be made by other users of FIFO</td>
</tr>
</tbody>
</table>

4.3.2.2.4 MWD

Each instance of this class that implements a client to a PC-based MWD. The MWD object takes care of all the behind the scenes socket communications with the PC.

The typical sequence of events for a MWD video session is as follows:
/ Create an MWD object
MWD* mwd = new MWD();
// Connect it to the right display
mwd->connect("somehost.somewhere", someport);
// Initialize the MWD
mwd->init(someformat, somebandcount, somebandIDs, somepixelcount);
// Loop outputs data and text
while (!done)
{
    // get some video
    ...
    // output line to MWD
    mwd->video(bandnumber, linenumber, video);
    // get a caption
    ...
    // output caption
    mwd->caption(linenumber, captiontext, captioncolor, captionalign);
}
// disconnect from the MWD, and delete object
mwd->disconnect();
delete mwd;
A description of the public methods of the MWD class is presented in the following table:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MWD()</td>
<td>Construct the MWD object, is initially not connected</td>
</tr>
<tr>
<td>MWDStatus connect(const string &amp;host, int port)</td>
<td>Connect to the MWD, must precede call to initialize. The host argument is a string, in either the “host.domain” format or in the “n.n.n.n” numeric IP address format. The port is the port number the MWD has been configured to listen on.</td>
</tr>
<tr>
<td>MWDStatus init(const string &amp;format, unsigned char band_count, const string &amp;bandIDs, short pixel_count)</td>
<td>Initialize the MWD. This method must be called before any other method except the constructor and initialize. The format is a string that is displayed on the MWD for data format. This format is also used to select a default setup for the MWD (bands displayed and any contrast enhancement in use). The band count is the number of bands that will be transferred to the MWD. The band IDs are a set of characters that will be displayed for each band (e.g., you could set the bands IDs to ‘ABC’ so the first band would be ‘A’). The pixel count are the number of pixels for each band. If the 4 parameters all match the last value used to initialize the MWD, this call is functionally equivalent to the intervalStart method below.</td>
</tr>
<tr>
<td>MWDStatus intervalStart(void)</td>
<td>Start a new interval on the MWD. Causes the MWD to reset its line numbering so that the next line that will be displayed is number 1. This call also discards any captions that have been stored at line numbers past the last video line received.</td>
</tr>
</tbody>
</table>
| MWDStatus video(unsigned char band_number, int line_number, const unsigned char* const videodata) | Send a line of video to the MWD. If the MWD is in autoscroll mode and the line number argument of this call is the highest yet seen since the last intervalStart or init call, the display will scroll to make this line the bottom visible line. Line numbers must be:  
- Greater than zero  
- Less than the highest line number seen plus some big constant (currently 4096 lines)  
- Greater than the last number seen for the band number  
The line numbers do not have to be consecutively higher than the last largest line number for the band. If they are not, the MWD will automatically fill the missing data with black lines. The MWD also supports BSQ formats where you send lines 1-N for one band, then lines 1-N for another band (bands do not need to be in ascending order). |
| MWDStatus caption(int line_number, const string &caption_text, const MWDColor color, const MWDAlignment alignment) | Send a caption to the MWD. The line number of the caption can be any line number greater than zero. Unlike the video command, the MWD will not scroll to make line number containing the caption visible. |
| MWDStatus gap(int size, const string &caption_text, const MWDColor color) | Insert a black gap after the highest video line that has been received by the MWD. A caption is displayed next to the gap. This method also starts a new interval like the startInterval method above. |
| MWDStatus disconnect(void) | Disconnect from the MWD |
In the above table there are three MWD specific types referenced. These are:

- **MWDColor** – a 24 bit color in a 3 byte structure, fields are red, green, and blue. There are eight predefined color constants you can use:
  - `coBlack`
  - `coRed`
  - `coGreen`
  - `coBlue`
  - `coYellow`
  - `coCyan`
  - `coPurple`
  - `coWhite`

- **MWDAlignment** – an enumerated type, defines which part of the text aligns with the specified line number:
  - `caTop`
  - `caMiddle`
  - `caBottom`

- **MWDStatus** – an enumerated type, defines error codes returned by the MWD client:
  - `msSUCCESS` Everything OK
  - `msCONNECT_ERROR` Error Connecting or Not Connected
  - `msPASSWORD_ERROR` Invalid connection password
  - `msSOCKET_ERROR` Socket I/O Error
  - `msINIT_ERROR` Error Initializing MWD
  - `msCAPTIONLENGTH_ERROR` Caption text empty or too long
  - `msGAPSIZE_ERROR` Gap size <= 0

### 4.3.2.2.5 FSD

Each instance of this class implements a client to a PC-based Frame Synchronizer Display (FSD). The FSD object takes care of all the behind the scenes socket communications with the FSD application on the PC.

The typical sequence of events for a FSD video session is as follows:
// Create an FSD object
FSD* fsd = new FSD();
// Connect it to the right display
fsd->connect("somehost.somewhere",someport);

// Loop outputs fsd updates
while (!done)
{
    FSDInfo fsd_info;
    // fill in info fields
    fsd_info.status = ... etc.
    // output update to FSD
    fsd->update(fsd_info);
}
// disconnect from the FSD, and delete object
fsd->disconnect();
delete fsd;

A description of the public methods of the FSD class is presented in the following table:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSD()</td>
<td>Construct the FSD object, is initially not connected</td>
</tr>
<tr>
<td>MWDStatus connect(</td>
<td>Connect to the FSD, must precede call to update. The host argument is a</td>
</tr>
<tr>
<td>const string &amp;host, int port)</td>
<td>string in either the &quot;host.domain&quot; format or in the &quot;n.n.n.n&quot; numeric IP</td>
</tr>
<tr>
<td>FSDStatus update(FSDInfo</td>
<td>Update the FSD with a new block of information</td>
</tr>
<tr>
<td>fsd_info)</td>
<td></td>
</tr>
<tr>
<td>FSDStatus disconnect(</td>
<td>Disconnect from the FSD</td>
</tr>
<tr>
<td>void)</td>
<td></td>
</tr>
<tr>
<td>~FSD(void)</td>
<td>Destructor, first disconnects if not disconnected, then cleans up</td>
</tr>
</tbody>
</table>

The FSDInfo block contains all of the information needed to update the display. The fields of this block are defined in FSD.hpp.

4.3.2.2.6 Ingest

The Ingest class is an abstract class that acquires blocks of raw data from a source and writes these blocks to a FIFO. Each actual ingest class (see the following subsections) must override these four methods:

// public abstract methods defined by Ingest
virtual bool prepare(void) = 0;
virtual bool go(void) = 0;
virtual bool isStopped(void) const = 0;
virtual bool stop(void) = 0;

The prepare method of any Ingest class generally creates a thread that will copy data from a source to the FIFO. This thread typically opens the data source, then waits on a
go flag. The go method of any Ingest class typically sets the go flag to allow the thread to start reading. The isStopped method checks for an indication that the copying operation has stopped. The stop method closes the data source and cleans up.

Each of the specific Ingest class descendents will also implement a static initialize method that reads the initialization file to determine the number of Ingest objects to create and the attributes of these objects. In most cases, the initialization file will contain a section like XXXIngest that looks something like this:

```plaintext
=XXXIngest
XXXIngestCount = N;
=XXXIngest1
# parameters for first ingest system
Key1 = Value1
Key2 = Value2
Etc.
=XXXIngest2
# parameters for second ingest system
Key1 = Value1
Key2 = Value2
Etc.
```

Note that this exact syntax is not enforced, just suggested. The initialize method will read the per object parameters from the initialization file, then call a constructor with these parameters to create each object, then insert the object into the Modules class.

4.3.2.2.6.1 RDCIngest

The RDCIngest class reads blocks of data from the RDC shared memory ring buffer and writes these blocks to a FIFO.

The static initialize method will read how many objects to be created from the initialization file. For L7 ingests, two objects will be constructed. For L5 ingests, one object. For each object, the initialization file will contain the name of the RDC ring buffer to use. The parameter will be loaded into a member of the RDCIngest object for use in attaching to the ring buffer.

The prepare method will create a run flag that is set, clear a stop request Boolean, then create and start a new thread object. It will then wait for the run flag to be reset before returning.

The run method of the thread object will attach to the RDC ring buffer and attach to the FIFO for the same string. In other words, for L7, the first RDCIngest will always attach to the first FIFO and the second to the second. After the attaches complete, the method will do a resetAndWait on the run flag. This will indicate that initialization has completed and allow the prepare method to exit.
The go method will set the run flag allowing the thread to start copying blocks of data from the ring buffer to the FIFO.

The thread’s run method will then loop, reading blocks from the ring buffer and writing the blocks to an output FIFO. When copying, any time the ring buffer is empty, the read call to the ring buffer block until data is available. Anytime the FIFO is full, a write call to the FIFO will block until the FIFO has room for more data. The run method will continue looping until either the ring buffer read method returns an end of stream indication or a stop request boolean is true. When either condition is encountered the thread will detach the RDC ring buffer, detach the FIFO, clear the run flag indicating completion, and then exit.

The isStopped method will read the state of the run flag and return its logical complement.

The Stop method will set the stop request boolean, then wait for the run flag to clear.

4.3.2.2.7 Frame Sync

The FrameSync class is an abstract class that acquires blocks of raw data from a FIFO, extracts frames of data from these blocks, and writes these frames to another FIFO. Each actual FrameSync class (see the following subsections) must override these four methods:

```cpp
// public abstract methods defined by Ingest
virtual bool prepare(void) = 0;
virtual bool go(void) = 0;
virtual bool isStopped(void) const = 0;
virtual bool stop(void) = 0;
```

The prepare method of any FrameSync class generally creates one or more threads that do the processing. One of these threads typically attaches to the source FIFO, then waits on a go flag. If there are additional threads they are synchronized to this first thread. The last thread attaches to the destination FIFO. The go method of any FrameSync class typically sets the go flag to allow the threads to start reading and processing. The isStopped method checks for an indication that the processing operation has stopped. The stop method closes the data source and cleans up.

Each of the specific FrameSync class descendants will also implement a static initialize method that reads the initialization file to determine the number of FrameSync objects to create and the attributes of these objects. In most cases, the initialization file will contain a section like XXXFrameSync that looks something like this:
Note that this exact syntax is not enforced, just suggested. The initialize method will read the per object parameters from the initialization file, then call a constructor with these parameters to create each object, then insert the object into the Modules class.

### 4.3.2.2.7.1 Landsat 7 Frame Sync

The Landsat 7 frame synchronizer is a CCSDS frame synchronizer. The L7 major frame (scan lines) are constructed by classes of the L7 Formatter. The raw incoming data stream is acquired from the input FIFO class object. Annotated CCSDS frames are output to the output FIFO class object.

The L7 Frame Sync is implemented as a FrameSync, Thread classes and is declared as:

```c
Class CCSDSFramer : public FrameSync, private Thread
```

The L7 Frame sync consists of the following main public methods, which will be found in the source files CCSDSFramer.cpp & CCSDSFramer.hpp.
Public:
static void Initialize( InitFile *inifile );

virtual SyncStatus getStatus( void );
virtual void getGroup1Name( string & );
virtual void getGroup2Name( string & );
virtual void getGroup1Labels( string &, string &, string &, string & );
virtual void getGroup2Labels( string &, string &, string &, string & );

virtual SyncStatus group1Status( long long int totals[5] );
virtual SyncStatus group2Status( long long int totals[5] );

virtual bool isStopped( void );
virtual bool prepare( void );
virtual bool go( void );
virtual bool stop( void );
virtual void abort( void );

Private:
int get( aCCSDSFrame *frame );
void maketable( void );
virtual int run( void );

The CCDSFramer constructor method is responsible for initialization of the member variables from the initialization file. The key initialization data member variables are the input FIFO, the minimum format change quality, and the maximum bit errors allowed in the CCSDS sync pattern. The CCDSFramer Initialize method is a static method that is responsible for reading the initialization file to determine if the CCSDS synchronization is selected, and if so, it instantiates two CCDSFramer objects and calls addFrameSync which is a Modules class static method, to add itself to the MWD system. The CCDSFramer prepare method attaches to the input raw FIFO and starts the thread that will process create the output frames. The CCDSFramer go method clears the runflag, which allows the processing on the thread to begin. The isStopped method checks for an indication that the copying operation has stopped. The stop method closes the data source and cleans up.

The getStatus method returns the primary status of the frame synchronizer; “Idle”, “Active”, “Wait”, “Lock” and “Search”. The methods getGroup1Name, getGroup2Name, getGroup1Labels, and getGroup2Labels return fixed strings that represent the names of the returned status values.
The methods group1Status and group2Status return arrays of status values. The CCSDS status values haven’t been determined, but normally include total frames, frames with bit errors in the sync pattern, detected bit slips, among others.

The run method is inherited from the Thread class and is used as the main routine of the CCSDSFramer. First it performs some initialization such as allocating memory buffers, creating the search table, and then waits for the run flag to be set using the Flag::waitForSignal() method which is described in the Flag section below. While the runflag continues to be set, the main loop creates the CCSDS frames.

The CCSDSFramer determines the ETM+ format of each decoded CCSDS frame, which is the only Landsat 7 specific logic in the CCSDSFramer class. Contained within the VCDU header is the ETM+ data format field, this format field is protected by an ECC. Since bit errors in telemetry is much more likely then a format change only a format and ECC field that has less then or equal number bit errors than an initialization set maximum (most likely zero) can cause the CCSDSFramer to change ETM+ formats. The annotated frame is then sent to the frame FIFO for the determined ETM+ format.

The run method will look like this:

```c++
Maketable;
runflag.waitForSignal;
current_format = 0;
format_changing = 0;
while ((runflag.isSignaled && get(ccsds_frame)));
{
    format = find_format(ccsds_frame, format_quality);
    if (format != current_format &&
        format_quality <= Min_change_quality)
        format_changing++;
    else
        format_changing = 0;
    if (format_changing > Change_threshold)
    {
        current_format = format;
        format_changing = 0;
    }
    if (current_format != 0)
        framefifo[current_format-1].write(ccsds_frame);
}
```

The CCSDS SearchTable is populated with the shift value for each bit position of the sync with one bit error in every position.
unsigned short mask, bits;
for ( int shift=0; shift<=16; shift++ )
{
    unsigned long long pattern = ccsds_sync;
    pattern <<=shift;                                    //sync pattern shifted by "shift" bits
    bits = (unsigned short) (pattern >>=48);            //top 16 bits moved to bottom 16 bits
    SearchTable[bits] = shift + 1;
    if( max_bad_bits > 0 )
    {
        for( j=0; j<16; j++ )
        {
            mask = 1 << j;
            if( SearchTable[(unsigned short)(bits^mask)] == 0 )
                SearchTable[(unsigned short)(bits^mask)] = shift + 1;
        }
    }
}

The private get method of the CCSDSFramer is where the bulk of the software frame synchronizer resides. The software frame sync key searching line of code from the get method looks like this:

```c
while( (search_ptr < stop_search) &&
    (SearchTable[*search_ptr]==0) ) search_ptr++ ;
```

Explanation: While the pointer into the search buffer is not too close to the end of the searching buffer and the 16bit word at the pointer is not a part of the CCSDS sync then increment the search pointer and try again. Only exiting from the one line loop when we need more raw data or a candidate sync pattern has been found. This simple approach allows the frame sync to fly through noise or other non-frame data with little CPU usage.

When more data is required, the unused data at the tail of the search buffer is copied to the head of the search buffer and new data is read in from the raw FIFO. The search pointer is set to the head of the buffer and the stop pointer is reset to the end of data in the buffer minus one frames length, which prevents a running out of buffer problem.

When a candidate sync pattern has been found, the shift value is extracted from the SearchTable, and converted from a 16bit-word offset to a 64bit-word offset. The area containing the full candidate sync pattern is shifted and XOR'ed with the known pattern. If the number of bits in error is less than or equal to the maximum allowed then the sync has been found else the search pointer is incremented and control is passed to the main searching "while".
The rest of the frame is shifted into alignment in the output frame buffer.

```c
for ( i=0; i<frame_size; i++ )
{
    buffer[i]  = *(pointer-1+i) << ( 64 – shift )  ;
    buffer[i] |= *(pointer+i)   >> shift  ;
}
```

The Pseudo Noise (PN) pattern that was applied to the frames before transmission is simply removed by XOR'ing with a stored copy of the PN pattern:

```c
for ( i=0; i<frame_size; i++ )
    buffer[i] ^= pn[i] ;
```

### 4.3.2.2.7.2 Landsat 4/5 TM Frame Sync

The Landsat 4/5 TM frame synchronizer constructs TM major frames (scan lines) from the raw incoming data stream. Incoming raw data is acquired from the input serial FIFO class object. Annotated major frames are output to the output frame FIFO class object.

#### Initialization parameters

The initialization parameters, retrieved via the initFile class, for the L45Framer are the following:

**Search_Buffer_Size** - The size in bytes of the allocated internal searching buffer. The value must be no smaller than the greater of two times the raw FIFO block sizes or two times the major frame size - e.g. TMR's major frame size is about 800 KB therefore the search_buffer_size can't be smaller than 1.6 MB. When zero, the default, the software will automatically set the buffer size to the smallest valid value, which is recommended. Note that although the searching buffer can be made very large, it should not be used for real-time data buffering. Must be an even value, or bus errors will occur.
MJF_Word_Match - Number of perfect 16bit words that must match in the major frame synchronization pattern during the searching phase. The default is three.

MJF_Word_Increment - How many 16bit words the software moves down the data stream on each check for a perfect 16bit word within the major frame synchronization pattern. The default is four.

MJF_Searching_Errors - The number of bit errors allowed in the 816 bit Major frame synchronization pattern. The default is 50.

MJF_Lock_Word_Match - During the Lock phase, the number of perfect 16bit words that must match in the next major frame synchronization pattern. The default is five.

MJF_Lock_Word_Increment – The number of 16bit words the software moves down the data stream on each check for the next major frame synchronization pattern. The default is one.

Beforestart_Timeout - The number of bytes of data to pass through the frame sync before the first frame is found before auto shutdown. A value of zero (0) disables this feature.

Afterstart_Timeout - The number of bytes of data to pass through the frame sync after the first frame is found before auto shutdown. A value of zero (0) disables this feature.

MNF_Flywheeling_Errors - The number of bits in error in the minor frame synchronization pattern before we start looking for bit slips. Minor frames with bit error values larger than this are considered to be flywheeled. The default is three.

Slip_Window_Back and Slip_Window_Ahead – The number of 16bit words to look back and ahead in the data stream window when looking for minor frame bit slips. The defaults are one.

Slip_Found - After a slip has been detected, this value is the number of bit errors that are allowed in newly located minor frame synchronization pattern before we perform a reshift of the data. The default is two.

Max_Slips - The maximum number of re-shifts of discovered bit slips that will be corrected in a single major frame. The default is five.

MNF_Lock_Loss_Trigger - Number of flywheeled minor frames in a row for which the minor frame synchronization pattern could not be found before a minor frame lock loss is declared.

Bit_File_Location – FOR TESTINGONLY - Saves a copy of the input bit stream to the specified directory with a file name of the date-time and a file type of *.bits*
Class Methods

The L4/5 TM Frame Sync is implemented as a FrameSync, Thread classes and is declared as:

```cpp
Class L45Framer : public FrameSync, private Thread
```

The L4/5 Frame sync consists of the following main public methods, which will be found in the source files L45Framer.cpp & L45Framer.hpp.

Public:
```cpp
static void initialize( InitFile *inifile );
static void writeVersion( ostream &vout ) ;
virtual SyncStatus getStatus( void ) ;
virtual void getGroup1Name( string & ) ;
virtual void getGroup2Name( string & ) ;
virtual void getGroup1Labels( string &, string &, string &, string & ) ;
virtual void getGroup2Labels( string &, string &, string &, string & ) ;
virtual SyncStatus group1Status( long long int totals[5] ) ;
virtual SyncStatus group2Status( long long int totals[5] ) ;
virtual bool isStopped( void ) ;
virtual bool prepare( void ) ;
virtual bool go( void ) ;
virtual bool stop( void ) ;
```

Private:
```cpp
L45Framer( InitFile *inifile, int sync_unit ) ;
int open( const char* bitfile ) ;
int close( void ) ;
int get( aCCSDSFrame *frame ) ;
void maketable( const char* bitfile ) ;
virtual int run( void ) ;
int get( unsigned char* out_frame, unsigned long long &grandtotalbit, 
unsigned long long &sync2sync, int &total_bad_bits, 
int &mnf_in_mjf, int &num_good_mnfs, int &num_bad_mnfs, 
int &num_slips, int &num_flywheeled ) ;
void statistics( TMRStatistics* tmrstats ) ;
int startFraming( void ) ;
int stopFraming( void ) ;
```
The L45Framer Initialize method is a static method that is responsible for reading the initialization file to determine if Landsat 4/5 TM synchronization is selected, and if so, it instantiates a L45Framer object and calls addFrameSync which is a Modules class static method, to add itself to the MWD system. The L45Framer constructor method is responsible for initialization of the member variables from the initialization file. All initialization data member variables have adequate default values. The L45Framer prepare method attaches to the input and output FIFOs and calls the private open method that creates the search tables and starts the thread that will process the output frames. The L45Framer go method calls the private startFraming method, which allows the processing on the thread to begin. The isStopped method checks for an indication that the copying operation has stopped. The stop method closes the data source and cleans up.

The getStatus method returns the primary status of the frame synchronizer; “Idle”, “Active”, “Wait”, “Lock” and “Search”. Where Idle and Active indicate the system hasn't started. The Wait state indicates that the software is waiting on more input data. The Search state indicates that the data is being processed but no frames are being found at the moment. The Lock state indicates that TM scan lines are being found and output.

The methods getGroupName, and getGroupName return fixed strings that are the names of the status groups. The Group 1 and 2 names are, respectively: “Major Frame Synchronizer”, ”Minor Frame Synchronizer”. The methods getGroup1Labels, and getGroup2Labels return fixed strings that represent the names of the status values that are returned by the methods group1Status and group2Status. The group1Status and group2Status return five values where the zeroth element is the total for the group, total major/minor frames and the other four are in the categories represented by the label names. These values are total values that increment continuously during the pass.

The getGroup1Labels and corresponding values:

- “Good” - Number of major frames with zero bit errors in the 816 bit major frame synchronization pattern.
- “Bit Errors” – Number of major frames with at least one bit in error in the 816 bit major frame synchronization pattern, but less bits in error than the maximum tolerance value which is set by the initialization file field of "MJF_Searching_Errors", which defaults to 50 when not provided.
- “MNF Losses” – Number of times minor frame lock was lost within the major frames. A minor frame lock loss is declared when the number of consecutive flywheeled minor frames exceeds the threshold set by the initialization file field of "MNF_Lock_Loss_Trigger", which defaults to 20 when not provided.
- “MJF Losses” – Number of major frame lock losses. A major frame lock loss is declared when subsequent major frame synchronization patterns are farther down the telemetry stream than expected.

The getGroup2Labels and corresponding values:
“Good” – Number of minor frames with zero bit errors in the 32 bit synchronization pattern.
“Bit Errors” – Number of minor frames with at least one bit in error in the 32 bit synchronization pattern.
“Bit Slips” – Number of bit slips that have been detected.
“Flywheeled” – Number of minor frames that have bit errors beyond the accepted tolerance. Frames are only counted as flywheeled when followed within the same major frame by acceptable minor frames, otherwise they would be indistinguishable from noise data.

The run method is inherited from the Thread class and is used as the primary processing routine of the L45Framer. First it performs some initialization such as attaching to the FIFOs, and then waits for the run flag to be set using the Flag::waitForSignal() method which is described in the Flag section. While the runflag continues to be set, the main loop creates the TM major frames, by calling the private get method repeatedly.

The private maketable method creates the searching tables. The major frame searching table, SearchTable is populated with the shift value for each bit position of the synchronization pattern. Valid shifts range from zero bit shift to 801 bit shift inclusive. The minor frame searching table, MnfSearchTable, has valid shifts ranging from zero bit shift to 16, and filled with two bit errors for each value. The tables are filled with shifts plus one to reserve zero in the table as a no match, this speeds processing on SGIs the test for zero is very fast. Allowing the frame synchronization process to go through noise easily.

The private get method of the L45Framer is where the bulk of the software frame synchronizer resides. The software frame sync key searching line of code from the get method looks like this:

```c
while( (search_ptr < stop_search) &&
       (SearchTable[*search_ptr]==0 ||
        SearchTable[*search_ptr]>min_val) )  search_ptr+= N_increment;
```

Explanation: While the pointer into the search buffer is not too close to the end of the searching buffer and the 16bit word at the pointer is not a part of the TM major frame synchronization pattern then increment the search pointer by the value supplied by the initialization parameter MJF_Word_Increment and try again. A second check is made to confirm that the thee indicated shift value is not too large, meaning the test for more matches is guaranteed to fail. Only exiting from the one line loop when we need more raw data or a candidate sync pattern has been found. This simple approach allows the frame sync to fly through noise or other non-frame data with little CPU usage.
When more data is required, the unused data at the tail of the search buffer is copied to
the head of the search buffer and new data is read in from the raw FIFO. The search
pointer is set to the head of the buffer and the stop pointer is reset to the end of data in
the buffer minus one frames length, which prevents a running out of buffer problem.

When a candidate sync pattern has been found, one match out of \(N\) has been found
in our major frame sync search. Where \(N\) is the value of the initialization parameter
MJF_Word_Match. Checks are then made to see if \(N-1\) more matches can be found
within the potential sync area. If these next checks confirm a major frame
synchronization pattern has been found a full bit-wise XOR'ed check is made to
determine the number of bits in error when compared to the expected pattern. When
the bits in error value is less than the value for the initialization parameter
MJF_Searching_Errors, then a major frame has been found.

All of the above checks are than made again at the expected location of the next major
frame synchronization pattern, at which point the general frame sync status is set to
“Lock”. The exact location of the second major frame synchronization pattern
determines the length of the major frame. When the next major frame can not be found
the major frame length is set to the maximum value.

The entire major frame is shifted into alignment in the output frame buffer.

```c
for ( i=0; i<frame_size; i++ )
{
    buffer[i] = *(pointer-1+i) << ( 64 – shift )  ;
    buffer[i] |= *(pointer+i)   >> shift  ;
}
```

Each of the minor frames within the major frame are then checked for the correct minor
frame synchronization pattern. The minor frame synchronization patterns will now be in
memory on word boundaries, making these checks a simple XOR without any shifting.

When minor frames are found with bit errors greater than the initialization parameter
MNF_Flywheeling_Errors, the minor frame is flywheeled. When the number of
flywheeled minor frames in-a-row exceeds the value of the initialization parameter of
MNF_Lock_Loss_Trigger then the minor frame lock loss statistics value is incremented.

When the size of the major frame is unknown due to not being able to find the next
major frame synchronization pattern and the minor frame lock loss has been triggered,
then the length of the major frame is reset to the point where the flywheeling started.

When three or more minor frames in-a-row are found to be flywheeled then a check is
made to confirm if a bit slip has occurred. The minor frame search buffer is used on the
adjacent words to locate the minor frame synchronization pattern. If found in a bit
slipped location the rest of the major frame from that point onward is then re-shifted.
Since this re-shifting is very time consuming a limit is put on the number of times a single major frame can be re-shifted, defined by the initialization parameter Max_Slips.

The Pseudo Noise (PN) pattern that was applied to the frames before transmission is then simply removed by XOR'ing with a stored copy of the PN pattern:

```
for ( i=0; i<frame_size; i++ )
    buffer[i] ^= pn[i];
```

4.3.2.2.8 Formatter

The Formatter class is an abstract class that acquires frames of data from a source FIFO, extracts and formats data for an MWD from these frames, and sends this information to the MWD. Each actual Formatter class (see the following subsections) must override these four methods:

```cpp
// public abstract methods defined by Ingest
virtual bool prepare(void) = 0;
virtual bool go(void) = 0;
virtual bool isStopped(void) const = 0;
virtual bool stop(void) = 0;
```

The `prepare` method of any Formatter class generally creates one or more threads that do the processing. One of these threads typically open the source FIFO, then waits on a go flag. If there are additional threads they are synchronized to this first thread. The last thread will send the output to an MWD (using an MWD object). The `go` method of any Formatter class typically sets the go flag to allow the threads to start reading and processing. The `isStopped` method checks for an indication that the processing operation has stopped. The `stop` method closes the data source and cleans up.

Each of the specific Formatter class descendents will also implement a static `initialize` method that reads the initialization file to determine the number of Formatter objects to create and the attributes of these objects. In most cases, the initialization file will contain a section like `XXXFormatter` that looks something like this:
=XXXFormatter
XXXFormatterCount = N;

=XXXFormatter1
# parameters for first Formatter system
Key1 = Value1
Key2 = Value2
Etc.

=XXXFormatter2
# parameters for second Formatter system
Key1 = Value1
Key2 = Value2
Etc.

Note that this exact syntax is not enforced, just suggested. The initialize method will read the per object parameters from the initialization file, then call a constructor with these parameters to create each object, then insert the object into the Modules class.

4.3.2.2.8.1 L7 Formatter

The L7 Formatter takes the input frames from the CCSDS frame sync via a frame FIFO and extracts the timecode and subsampled video lines and sends them to the MWD. It also determines interval starts, which are displayed on the MWD as well as dropped lines which show up on the MWD as blank lines.

The L7 Formatter is implemented as a Formatter, Thread class and is declared as:

Class FormatterL7 : public Formatter, private Thread

The L7 Formatter consists of the following main methods, which are found in the source files FormatterL7.cpp & FormatterL7.hpp.
The FormatterL7 constructor method is responsible for initialization of the member variables from the initialization file. The key initialization data member variables are the MWD subsampling factors (x,y), and the detector/band offsets for the L7 ETM+ video lines. The FormatterL7 Initialize method is a static method that is responsible for reading the initialization file to determine if the L7 formatter is selected, and if so, it instantiates two FormatterL7 objects and for each calls addFormatter which is a Modules class static method, to add itself to the MWD Formatter system. The FormatterL7 prepare method attaches to the frame FIFO from the CCSDS frame sync and starts the thread that will process the L7 frames. The FormatterL7 go method clears the runflag, which allows the processing on the FormatterL7 thread to begin.
The `isStopped` method checks for an indication that the copying operation has stopped. The `stop` method closes the data source and cleans up.

The `run` method is inherited from the `Thread` class and is used as the main routine of the L7 Formatter. First it performs some initialization such as allocating some memory buffers and then waits for the run flag to be set using the `Flag::waitForSignal()` method which is described in the `Flag` section below. While the runflag continues to be set, the main loop calls the get method of the L7MajorFrame object to retrieve an annotated L7 scan line if the timecode is reliable, as determined by the timecode quality value, it can be used to compute the number of dropped major frames (if any). If dropped major frames are encountered that are less than the threshold value, this number is used to increment the interval line counter by the number of dropped frames. Incrementing the interval line count will allow the dropouts to appear on the MWD as filled lines. If the number of dropped frames was greater than the threshold or negative, a new interval is encountered. When a new interval is encountered, the methods `endInterval` and `startInterval` are called. Finally the `writeFrame` method is called to perform the L7 ETM+ decomutation, subsampling and to write the video to the MWD.

```
PDL for the FormatterL7::run( void ) method

Initialize, allocate memory buffers

Construct L7MajorFrame object

RunFlag.waitForSignal() Wait for the runflag

While ( runflag is signaled )

    Get L7 major frame, break on EOD
    Compute number_of_fill_frames
    If ( number_of_fill_frames < 0 or number_fill_frames > threshold ) then
        endInterval()
        startInterval()
        else if ( number_of_fill_frames > 0 )then
        increment interval line number ( to display gap on MWD )
        endif
        writeFrame() to output subsampled video to MWD

end While
```

The `startInterval` method initializes the MWD by calling the `mwd` class `gap`, `init`, and `intervalStart` methods. It also displays the interval start timecode using the `mwd` class `caption` method. The `endInterval` method displays the interval stop timecode using the `mwd` class `caption` method.
The `writeFrame` method performs the L7 ETM+ frame decomutation, which extracts the raw image lines from a L7 ETM+ major frame. The ETM+ major frame is directly decomutated into subsampled image lines for the MWD. The MWD class `caption` method to display the timecode for every Nth image line where N is some programmable number such as 100 such that the spacing is sufficient so that the captions do not overwrite one another. Next the decomutation, subsampling and MWD output is performed as is described in the following PDL. To make the MWD output of all bands the same size, the subsampling rate varies with band size. The subsampling rate is based on the standard reflective bands, for band 6 is half of the standard, and the subsampling rate for band 8 (PAN) is double that of standard.

PDL for the ETM+ decomutation in the FormatterL7::writeFrame method

```
Loop through all bands
  Loop through all detectors for each band
    If ( current line/detector has been selected based on y-direction subsampling ) then
      If ( initfile bumpermode flag set ) then
        Compute bumper mode offsets into the major frame for the current band/detector
      else
        Compute normal mode offsets into the major frame for the current band/detector
      endif
    if ( scan_direction is forward )
      Loop through the Major frame extracting subsampled image data for the detector/band into a buffer in the forward direction ( x-subsampled )
    else
      Loop through the Major frame extracting subsampled image data for the detector/band into a buffer in the reverse direction ( x-subsampled )
    endif
    use the MWD class VIDEO method to write current band image data to MWD
  endif
end detector loop
end band loop

every N lines use MWD::Caption to display timecode on the MWD
```

L7 major frames are The Landsat 7 major frame class, L7MajorFrame, builds and annotates L7 major frames (ETM+ scans). CCSDS frames are acquired as input from the frame FIFO. The L7MajorFrame class consists of the following methods.
The constructor allocates the correct input frame FIFO for the assigned format.

Each CCSDS frame is examined for its relationship to the L7 scan line start (SLS), CCSDS frames before the first SLS are discarded. Each CCSDS frame contains the minor frame counter and byte pointer within that minor frame. The next pointer and previous pointer are calculatable from the value of the current pointer in a highly defined manner. Each pair of CCSDS frames is examined for the case where the minor frame counter resets and/or the byte pointer value is not as expected, the SLS may be nearby. The expected beginning of the SLS is calculated from the minor frame counter and byte pointer. The SLS is a minor frame containing 40 bytes of all bits set followed by 40 bytes of all bits clear. The SLS may be wholly contained in either of the two CCSDS frames, or even span the CCSDS frame boundary. The "extra zero" problem, noted in the L7 DFCB Vol. 4 section 3.2.6.2.3.2.4, is detected when the SLS is located starting one byte earlier than expected.

The L7 scan line is built by loading each successive CCSDS frame's minor frame data into the major frame. The loading continues until the next SLS is encountered or the maximum number of minor frames in a major frame is reached. When CCSDS frames have been lost, fill minor frames are loaded into the scan line, based on the channel counter and pointer fields. Because the loading usually stops when the next SLS is encountered, subsequent calls to the "get" routine find the SLS immediately.

The scan direction, multiplexer identifier and the gain settings for each imagery band are present as a single bit in each CCSDS frame. The bytes containing these bit fields are saved from, an initialization set number of consecutive CCSDS frames. Each of these bits is then majority voted determining their values with a high degree of reliability.

The utility routine "get_time" is called to populate the annotated L7 scan line with the timecode of the scan and the bit error quality of the timecode. The timecode of the scan line is extracted from minor frames 2 through 5, into both a string version of the time and a double floating value stored in the annotated major frame structure. The timecode fields in the minor frame are 40 bit fields, all set or all clear. Each 40 bit field represents a single bit of a digit of the timecode. Each of the digits is 2 to 4 bits in size.
The total number of bits not in consensus is saved for determining the timecodes reliability.

The utility routine “find_EOL” returns the number of minor frames until the end of line code. The end of line pattern is 40 bits clear followed by 40 bits set repeating for 2 minor frames, exclusive of the 5 bytes and the end of each minor frame representing the band 6 areas. Each minor frame within, an initialization set, window is checked for the end of line pattern. When the number of bits different from what is expected is less than, an initialization set, number the end of line has been found.

```c
void find_EOL( aL7MajorFrame* mjf )
{
    int value ;
    int cur_mnf = Min_EOL_mnf ;
    mjf->counted_mnf = 0 ;
    while( cur_mnf < Max_EOL_mnf && !mjf->counted_mnf ) {
        cur_mnf++ ;
        value = bits_on( mjf->major_frame+(cur_mnf*L7mnfSize), 20 )
            + bits_off( mjf->major_frame+(cur_mnf*L7mnfSize)+20, 20 )
            + bits_on( mjf->major_frame+(cur_mnf*L7mnfSize)+40, 20 )
            + bits_off( mjf->major_frame+(cur_mnf*L7mnfSize)+60, 20 ) ;
        if( value < 40 ) {
            mjf->counted_mnf = cur_mnf ;
            total = value ;
        }
    }
    if( mjf->counted_mnf ) {
        value = bits_on( mjf->major_frame+(cur_mnf*L7mnfSize), 20 )
            + bits_off( mjf->major_frame+(cur_mnf*L7mnfSize)+20, 20 )
            + bits_on( mjf->major_frame+(cur_mnf*L7mnfSize)+40, 20 )
            + bits_off( mjf->major_frame+(cur_mnf*L7mnfSize)+60, 20 ) ;
        total += value ;
        mjf->eol_flag = 0 ;
    } else {
        cerr << "End-of-Scan code not found." << endl ;
        mjf->eol_flag = 1 ;
        mjf->counted_mnf = (Max_EOL_mnf-Min_EOL_mnf)/2 + Min_EOL_mnf ;
        total = 0 ;
    }
    mjf->counted_mnf_errors = total ;
    return ;
}
```

The base utility routines bits_on and bits_off return the number of bits set or clear starting at the given pointer for the given number of bytes. The “value” in each of the statements above would be zero in perfect data.

4.3.2.2.8.2 L5 Formatter

The L5 Formatter takes the input frames from the L5 frame sync via a FIFO and extracts the timecode and subsampled video lines and sends them to the MWD. It also determines interval starts, which are displayed on the MWD as well as dropped lines which show up on the MWD as blank lines.
The L5 Formatter is implemented as a **Formatter Thread** class and is declared as:

```plaintext
Class FormatterL5 : public Formatter, private Thread
```

The L5 Formatter consists of the following main methods, which are found in the source files FormatterL5.cpp & FormatterL5.hpp.

```plaintext
FormatterL5( InitFile* );                     // constructor
virtual bool FormatterL5::prepare(void);     // overrides base class Formatter
virtual bool FormatterL5::go(void);          // overrides base class Formatter
virtual bool FormatterL5::isStopped(void) const;  // overrides base class Formatter
virtual bool FormatterL5::stop(void);        // overrides base class Formatter
virtual int  FormatterL5::run(void);         // overrides base class Thread

void startInterval( scanTime *time, Interval* &ival );
void endInterval( scanTime *time, Interval* &ival, int lines, int sublines, int frames ) ;
void writeFrame( scanTime *time, double actual_time, int &line_num, int &sub_num, int lines, int sublines, int frames ) ;
```

The **FormatterL5 constructor** method is responsible for initialization of the member variables from the initialization file. The key initialization data member variables are the MWD subsampling factors (x,y), the bumper mode flag and the detector/band offsets for the L5 TMR video lines. The **FormatterL5 Initialize** method is a static method that is responsible for reading the initialization file to determine if the L5 formatter is selected, and if so, it instantiates a FormatterL5 object and calls `addFormatter` which is a Modules class static method, to add itself to the MWD Formatter system. The **FormatterL5 prepare** method attaches to the FIFO from the L5 framesync and starts the thread that will process the L5 frames. The **FormatterL5 go** method clears the runflag, which allows the processing on the FormatterL5 thread to begin. The **isStopped** method checks for an indication that the copying operation has stopped. The **stop** method closes the data source and cleans up.

The **run** method is inherited from the Thread class and is used as the main routine of the L5 Formatter. First it performs some initialization such as allocating some memory buffers and then waits for the run flag to be set using the **Flag::waitForSignal()** method which is described in the **Flag** section below. While the runflag continues to be set, the main loop reads a block of data (a L5 TMR major frame) from the FIFO. For each frame the L5 timecode is computed using the scanTime class method **compute**. Next it calls the **getVoteErrors** method to determine the worst case number of time code voting errors from all of the components that make up the timecode. This worst case voting
values is used to determine if the timecode is reliable. If the timecode is reliable, it can be used to compute the number of dropped major frames (if any). If dropped major frames are encountered that are less than the threshold value, this number is used to increment the interval line counter by the number of dropped frames. Incrementing the interval line count will allow the dropouts to appear on the MWD as filled lines. If the number of dropped frames was greater than the threshold or negative, a new interval is encountered. When a new interval is encountered, the methods `endInterval` and `startInterval` are called. Finally the `writeFrame` method is called to perform the L5 TMR decomutation, subsampling and to write the video to the MWD.

```
PDL for the FormatterL5::run( void ) method

Initialize, allocate memory buffers

RunFlag.waitForSignal() Wait for the runflag

While ( runflag is signaled )

    Read a major frame from the FIFO
    Compute the timecode
    Get the max voting errors
    Compute number_of_fill_frames
    If ( number_of_fill_frames < 0 or number_fill_frames > threshold ) then
        endInterval()
        startInterval()
    else if ( number_of_fill_frames > 0 ) then
        increment interval line number ( to display gap on MWD )
    endif

    writeFrame() to output subsampled video to MWD

end While
```

The `startInterval` method initializes the MWD by calling the `mwd` class `gap`, `init`, and `intervalStart` methods. It also displays the interval start timecode using the `mwd` class `caption` method. The `endInterval` method displays the interval stop timecode using the `mwd` class `caption` method.

The `writeFrame` method performs the L5 TMR frame decomutation, which extracts the raw image lines from a L5 TMR major frame. The TMR major frame is directly decomutated into subsampled image lines for the MWD. The utility class `endScan` method `findEndScan` is called. This is necessary to extract the scan direction. The `MWD` class `caption` method to display the timecode for every Nth image line where N is some programmable number such as 100 such that the spacing is sufficient so that the captions do not overwrite one another. Next the decomutation, subsampling and MWD output is performed as is described in the following PDL.
The L5 Formatter uses the following utility classes scanTime and endScan, which are found in the source files L5TMR_utility.cpp & L5TMR_utility.hpp. The key methods from these classes are described here:

class endScan {
    endScan(int start=6300, int stop=6340);
    int findEndScan(int frame_num, L45FIFOFrame *frame);
    int getDir()  { return dir;};
    void compute(int frame_num, unsigned char* data);
};
The `endScan::endScan()` constructor initializes the current object with the start and stop minor frame for the search. The `findEndScan` method searches for the end scan pattern and then calls the `compute` method. The `compute` method performs the extraction of all of the fields within the L5 TMR endscan and stores them in the data members to be retrieved when called for. The `getDir` method returns the `int` scan direction data member `dir`. There are other data members and methods in the L5 `endScan` class are not used by the MWD formatter.

```cpp
class scanTime {
    double getTime(){return time;};
    string& getMtime(){return mtime;};
    getVoteError(){return worst_vote_error;};
    private:
    double time;
    string mtime;
};
```

The `compute` method extracts all of the data bits that make up the various components of the L5 TMR timecode. This is done by looping through the 6 minor frames that make up the timecode and extracting the groups of 48 copies of each data bit and majority voting to compute the value of each bit. For each of the majority voted bits, the worst-case majority vote error count is saved. The majority voted bits are assembled into the following components:

- Tens of days
- Tens of hours
- Tens of minutes
- Tens of seconds
- Hundreds of Milliseconds
- Single Milliseconds
- Spacecraft ID
- Hundreds of days
- Single Days
- Single Hours
- Single Minutes
- Single Seconds
- Tens of Milliseconds
- Sixteenths of Milliseconds

When all the bits are assembled, the various components parts of the timecode are assembled as individual integer values. The component integers are assembled to produce a numeric timecode in decimal seconds that is stored in a double precision floating point data member `time` and as an ascii string which is stored in the string data member `mtime`. The method `getTime` returns the numeric time as a `double` and the method `getMtime` returns the ascii timecode as a `string`. 
4.3.2.3 Library Classes

4.3.2.3.1 Initfile

The Initfile class is a wrapper object around an initialization file that contains standardized methods for dealing with these files. Each instance of this class contains an initialization file that is held in memory. The contents of the initialization file may be interrogated, updated, and saved back to disk through the methods of this class.

The normal use for opening, reading, and optionally writing initialization files with this class is something like the following:

```cpp
InitFile initfile("~/mydir/initfile.ini");
String s = initfile.getString("MySection", "MyKeyword");
int n = initfile.getInteger("MySection", "MyInteger", -1);
string s = initfile.getString("MySection", "MyString", "Default");
// etc.
initfile->save(); // Optional
```
The format of an initialization file is generally as follows:

```plaintext
# Default section
#
keyword0_1=value0_1
keyword0_2=value0_2
...
keyword0_N=value0_N

=section1
#
# Section 1 Comments
#
keyword1_1=value1_1
keyword1_2=value1_2
...
keyword1_N=value1_N

=section2
#
# Section 2 Comments
#
keyword2_1=value2_1
keyword2_2=value2_2
...
keyword2_N=value2_N

...

=sectionM
#
# Section M Comments
#
keywordM_1=valueM_1
keywordM_2=valueM_2
...
keywordM_N=valueM_N
```

Initialization files have the following semantic rules:

**<RULE 1>**

Leading blank characters are trimmed from all lines before the line is processed. So, the first character of a line is the first non-blank character of the line.

**<RULE 2>**

Any line starting with a '#' or a '!' is a comment.
<RULE 3>

Any line starting with '=' is the beginning of a section. The section name should immediately follow and be made up of any combination of letters including embedded spaces. Leading and trailing white space are not considered. Some valid examples:

```
=DRIVES       translates to "DRIVES"
= CURRENT DRIVES translates to "CURRENT DRIVES"
```

<RULE 4>

Any line starting with anything else and containing a "=" is considered to be a keyword line. These lines start with a keyword that is followed by an '=' and then optionally end with a value. Like section names, keywords can be made up of any string of characters, except keywords may not contain '=' and may not start with "#" or '!'. For example:

```
DESCRIPTION=A Test Program
DRIVE COUNT=2
DRIVE1= DLT10
DRIVE2 = DLT18
```

Note that for values:
- Everything past leading whitespace (tabs,blanks) and before trailing whitespace is part of the value
- Numeric values are stored as strings

<RULE 5>

Any line that doesn't start with '#' or '!' or that doesn't contain a '=' is invalid. However, the initfile routines currently handles lines it doesn't understand like comments.

<Other Notes>

Everything between the top of the file and the first section header is considered to be the default section. It can be accessed by using an empty section name. If the default section is deleted, everything above the first section header is deleted, including comments. If the default section is cleared, everything above the first section header is removed except comment lines.

Everything between a =section<\n> and the next =section<\n+1> belongs to section<\n>. If section<\n> is deleted, all lines in between are deleted, including all comments. If section<\n> is cleared, all keyword lines will be deleted, but comments with the section will be left unchanged. When a section is created, it is created at the end of the file, with one preceding blank line, and with no comments

When a keyword is created, it is created at the end of the section, before any trailing blank lines. When a keyword line is replaced, it is replaced in the same position.
For an initialization file where keywords may be added by code, it is strongly suggested that you follow one or both of these conventions:

A) Do not interleave comment lines with keyword lines
B) Place anticipated keywords with empty values in the file so that only updates occur, leaving the comment structure untouched.

Section name and keyword matching is case-insensitive.

If the same section name occurs twice in a file, only the first section will be used. Similarly, if the same keyword name occurs twice in a section, only the first keyword will be used.

The complete synopsis of public entry points follows:

```c
InitFile(void);
InitFile(string pathname);
-InitFile(void);
bool load(string pathname);
bool isLoaded(void);
bool isModified(void);
bool containsSection(const string &section);
bool containsKeyword(const string &section, const string &keyword);
string getName();
string getString(const string &section, const string &keyword);
string getString(const string &section, const string &keyword, const string &defValue);
void setString(const string &section, const string &keyword, const string &value);
int getInteger(const string &section, const string &keyword, int defValue);
void setInteger(const string &section, const string &keyword, int value);
long long int getLong(const string &section, const string &keyword, long long int defValue);
void setLong(const string &section, const string &keyword, long long int value);
float getFloat(const string &section, const string &keyword, float defValue);
void setFloat(const string &section, const string &keyword, float value);
double getDouble(const string &section, const string &keyword, double defValue);
void setDouble(const string &section, const string &keyword, double value);
void setBoolean(const string &section, const string &keyword, bool value);
void clearSection(const string &section);
void deleteSection(const string &section);
void deleteKeyword (const string &section, const string &keyword);
bool save();
bool save(string pathname);
```

4.3.2.3.2 Log

The Log class provides a standard set of routines for opening, writing, and closing log files. This logging system is designed to allow multiple processes created using the Thread classes to log messages to a common file. The methods of this class are all static, meaning the individual logging routines can use Log::xxx() to access the log without having to include a global log object variable somewhere. Unfortunately the downside of this is only a single log file can be managed by these routines.

Each message is logged to the file as a single line in the following standard form:
YYYY-MM-DD HH:MM:SS | <pid> | <source> | <Method> | <Level> | <Msg>

For example:

2002-03-28 11:39:34 | 138970 | lts.cpp | LTSInitializer | INFO | Initializing main modules

The date, time, and process ID (pid) are automatically inserted into the message at the time the message is logged. The remaining fields must be provided in the call to log the message. An example of using the generalized method for logging a message is:

```
Log::message(LogINFO, __FILE__, "LTSInitializer", "Initializing main modules");
```

For convenience, individual shorthand methods for each level are provided:

```
Log::info(__FILE__, "LTSInitializer", "Initializing main modules");
```

The logging routines also allow the minimum logging level to be dynamically set. Log levels, in ascending order of priority are:

- **DEBUG** Diagnostic information for developers/maintainers
- **INFO** Informative messages for significant events
- **WARNING** Messages that indicate minor problems
- **ERROR** Messages that indicate major problems
- **FATAL** Messages that indicate fatal problems (process exits)

The following table presents a synopsis of the methods are available for using in this class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static int open(const string &amp;logpath)</td>
<td>Open the log file stream. If logpath is an empty string, if the file cannot be opened, or if this routine is not explicitly called the log routines will direct log output to the standard output stream. Minimum_level will establish the minimum priority log message that will be output. If this method is not called all messages will be logged.</td>
</tr>
<tr>
<td>Int reopen(const string &amp;logpath)</td>
<td>Closes the current logfile, then opens a new one</td>
</tr>
<tr>
<td>Int message(LogLevel level, const string &amp;source, const string &amp;routine, const string &amp;message)</td>
<td>General version of message logging method, log a message using specified level, source file name, routine name, and message.</td>
</tr>
<tr>
<td>Int debug(const string &amp;source, const string &amp;routine, const string &amp;message)</td>
<td>Log message a level implied by name of method called. All 5 methods call the message method to actually log the message.</td>
</tr>
</tbody>
</table>
4.3.2.3.3 Thread

This source contains a set of classes that can be used to create and synchronize sproc based threads. This implementation abstracts the underlying implementation of the threads being used (sprocs or pthreads) so that it may be possible to switch the type in use in the future without rewriting a lot of code. There are three classes currently defined in this source:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>An abstract thread class that uses sprocs to create threads</td>
</tr>
<tr>
<td>Gate</td>
<td>Implements a simplified mutex</td>
</tr>
<tr>
<td>Flag</td>
<td>Implements a simplified condition variable</td>
</tr>
</tbody>
</table>

4.3.2.3.3.1 Abstract Thread

An abstract thread class encapsulates the complex code needed to construct and manage threads into a tidy simple-to-use object. To create a thread one must create a subclass of the Abstract Thread class and override the run method as follows:

```cpp
class MyThread: public Thread
{
    virtual int run(void);
}

int MyThread::run(void)
{
    int exit_status; // exit status for process
    ...code executed by thread goes here, thread exits on return
    return exit_status;
}
```
Then, an instance of a thread object should be created. The thread will not run until started using the \textit{start} method.

\begin{verbatim}
MyThread myThread; // Thread object

int main()
{
    myThread.start(); // start the thread
    ...etc
}
\end{verbatim}

There are many times when it is handy to determine the current thread. You can do this by calling the static method:

\begin{verbatim}
Thread* currentThread = Thread::getCurrentThread();
\end{verbatim}

When the thread classes are first used an internal thread object is constructed that is a representation of the main thread. So, the above call even works from the main thread.

\textbf{WARNING}, you should not combine the use of any variant of \texttt{fork()} with the thread methods. It is much safer to use the \texttt{executeAndWait} methods of this class.
The following table contains a synopsis of the thread methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread(void)</td>
<td>Construct a thread object, thread not created until start called</td>
</tr>
<tr>
<td>int start(void)</td>
<td>Start the thread, creates a new thread, calls run(void) as method of object. Returns PID of newly created thread or error code. Files handles are implicitly shared.</td>
</tr>
<tr>
<td>int start(bool shared_files)</td>
<td>As above, but can be used to explicitly not share files.</td>
</tr>
<tr>
<td>pid_t getThreadID(void) const</td>
<td>Gets the ID of the thread (if running)</td>
</tr>
<tr>
<td>ThreadStatus getThreadStatus(void)</td>
<td>Sets the status of the thread, one of tsACTIVE, tsIDLE, tsDONE, or tsFAILED</td>
</tr>
<tr>
<td>bool setSignalHandling(int sig, SigHandling sig_handling)</td>
<td>Set signal handling for signal to shIGNORE or shDEFAULT. Used to ignore certain signals, such as SIGPIPE</td>
</tr>
<tr>
<td>static Thread* currentThread(void)</td>
<td>Get a pointer to the current thread, can be main thread. Pointer can then be used to get status or ID of thread.</td>
</tr>
<tr>
<td>static int executeAndWait(const string &amp;command, const string &amp;name, int nice incr = 0)</td>
<td>These calls are used to fork() and exec() a detached process without adversely impacting other processing that is occurring. It is strongly recommended that these calls be used instead of directly calling the Unix methods. All variants accept a command line, a process name, and a nice increment. The second version also accept names for files to which the text from standard out and standard error should be appended. The third version accepts everything in the second version along with the name of a directory to set as default before executing the process</td>
</tr>
<tr>
<td>static int executeAndWait(const string &amp;command, const string &amp;name, const string &amp;std_out, const string &amp;std_err, int nice incr = 0)</td>
<td></td>
</tr>
<tr>
<td>static int executeAndWait(const string &amp;command, const string &amp;name, const string &amp;directory, const string &amp;std_out, const string &amp;std_err, int nice incr = 0)</td>
<td></td>
</tr>
<tr>
<td>~Thread(void);</td>
<td>Cleanup a thread object. Note that as of this implementation, this method does not cancel the running thread and therefore should not be called until the thread exits.</td>
</tr>
</tbody>
</table>

### 4.3.2.3.3.2 Gate

Gates are objects that may be used to synchronize access to a critical section of code such as manipulating a linked list or a collection that is shared by multiple threads. One should use a gate object as follows:

```pseudocode
Gate gate; // a Gate, initially open
gate.enter(); // Enter critical section of code
...non-reentrant code here

gate.leave(); // Leave critical section of code
```
The following table presents a synopsis of the Gate methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gate(void)</td>
<td>Construct an initially open gate</td>
</tr>
<tr>
<td>void enter(void)</td>
<td>Enter gate, will wait if some other thread has already entered but not yet left gate. enter and leave method calls must be balanced.</td>
</tr>
<tr>
<td>void leave(void)</td>
<td>Leave gate, gate not really left until a leave is called for every enter. Attempt to leave a gate without entering will cause deliberate core dump via ::abort().</td>
</tr>
<tr>
<td>void yield(void)</td>
<td>Momentarily release gate then reacquire. Allows other threads to sneak in and get some time for long-running locked operations.</td>
</tr>
<tr>
<td>bool isLocked(void)</td>
<td>Returns true if gate has been entered, false otherwise.</td>
</tr>
<tr>
<td>~Gate(void)</td>
<td>Releases resources in use by gate, should not be called if other threads are still using the gate</td>
</tr>
</tbody>
</table>

4.3.2.3.3 Flag

Flags are used to wait for some other thread to signal a condition. The simplest use of a flag object follows:

```c
Flag flag;              // A flag, initially reset

// <code in thread1>
flag.waitForSignal();   // Wait for other thread to set flag

// <code in thread2>
flag.signal();          // Set flag to unblock thread 1
```

Flag objects can also be used to implement a rendezvous between two processes:

```c
// <code in thread1>
flag.waitForSignal();   // Wait for permission to run code
...code in rendezvous... // Actions thread 2 is waiting on
flag.reset();           // Indicate rendezvous complete

// <code in thread2>
flag.signalAndWait();   // Set flag to unblock thread 1
// Wait for rendezvous to complete
```
The following table presents a synopsis of the Flag methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag(void)</td>
<td>Construct an initially reset flag</td>
</tr>
<tr>
<td>bool signal(void)</td>
<td>Set flag, returns previous state of flag. All processes waiting for flag to be set are unblocked</td>
</tr>
<tr>
<td>bool reset(void)</td>
<td>Reset flag, returns previous state of flag. All processes waiting for flag to be reset are unblocked</td>
</tr>
<tr>
<td>bool waitForSignal(void)</td>
<td>Block thread until flag is set. Returns state of flag before call</td>
</tr>
<tr>
<td>bool waitForReset(void)</td>
<td>Block thread until flag is reset. Returns state of flag before call</td>
</tr>
<tr>
<td>bool resetAndWait(void)</td>
<td>Reset flag then wait for it to be set. Returns state of flag before call</td>
</tr>
<tr>
<td>bool signalAndWait(void)</td>
<td>Set flag then wait for it to be reset. Returns state of flag before call</td>
</tr>
<tr>
<td>bool isSignaled(void)</td>
<td>Returns state of flag without waiting.</td>
</tr>
<tr>
<td>~Flag(void)</td>
<td>Releases resources in use by flag, should not be called if other threads are still using the flag</td>
</tr>
</tbody>
</table>

4.3.2.3.4 Sockets

This source implements four classes that can be used to manage and use TCP/IP sockets:

- **BaseSocket** A wrapper around a socket descriptor, base of all others
- **Socket** Implements methods common to all end-point sockets
- **ServerSocket** Implements a socket to which clients can connect
- **ClientSocket** Implements a client socket

To use these classes, the server process creates a server socket, then waits for the client routine to connect as follows:

```c
ServerSocket ss = new ServerSocket(APORT);
if (ss->getStatus() == ssSUCCESS)
{
    Socket as = ss->accept();
    if (as->getStatus() == ssSUCCESS)
        ...
    // (or)
    ServerSocket ss = new ServerSocket();
    ss->open(APORT);
    if (ss->getStatus() == ssSUCCESS)
        {
            Socket as = ss->accept();
            ...
```
The client process then connects to the server and send a message to it as follows:

```java
ClientSocket cs = new ClientSocket("Server_IP", APORT);
if (cs->getStatus() == ssSUCCESS)
{
    status = cs.writeBuffer(AMessage, sizeof(AMessage));
    ...
    // (or)
    //
    ClientSocket cs = new ClientSocket();
    cs->connect("Server_IP", APORT);
    if (cs->getStatus() == ssSUCCESS)
    {
        status = cs.writeBuffer(AMessage, sizeof(AMessage));
        ...
}
```

The server process then reads the message, responds, and closes the connection as follows:

```java
while (processing) {
    status = as->readBuffer(AMessage, sizeof(AMessage));
    ... deal with message ...
    status = as->writeBuffer(AResponse, sizeof(AResponse));
}
as->close();
```

Finally, the client reads the response and closes the connection from its end as follows:

```java
while (processing) {
    status = cs->read(AResponse, sizeof(AResponse));
    ... Construct next message
    .status = cs->write(AMessage, sizeof(AMessage));
}
// cs->close();
```

The server process can also close the server socket if desired:

```java
ss->close();
```
The socket can be used for multiple reads and writes if desired. There are a number of entry points that can be used to read and write other types of data. These methods insure that the data is written to the socket in the correct byte order, regardless of host machine "endianism":

```
// read routines
b = socket->readByte();     // 8 bit unsigned
s = socket->readShort();    // 16 bit signed
i = socket->readInteger();  // 32 bit signed
l = socket->readLong();     // 64 bit signed
socket->readString(t);      // C++ 'string'

// write routines
socket->writeByte(b);       // 8 bit unsigned
socket->writeShort(s);      // 16 bit signed
socket->writeInteger(i);    // 32 bit signed
socket->writeLong(l);       // 64 bit signed
socket->writeString(t);     // C++ 'string'
```

The status of a socket can be checked at any time:

```
SocketStatus socket_status = socket->getStatus();
```

Any status other than ssSUCCESS for an open socket indicates an error. ssCLOSED is normal for a closed socket. Any time an error occurs on a socket, the socket is automatically closed.

To provide a little security for sockets, both the server and the client process can simultaneously use the secureCheck method. It is recommended that this be done immediately after startup. Immediately after the accept by the server and the connect by the client, the following call should be made:

```
asocket->secureCheck("Some Password String");
```

The passwords on both side must match. When called, each half of the connection will:
1. send a 16 byte random sequence of bytes to the opposite side.
2. receive and encrypt the 16 byte sequence using the password
3. send the encrypted 16 bytes back
4. compare the encrypted result received with what it should be
5. set ssSECURE_ERROR and close the socket if it doesn't match

The encryption used is a relatively simplistic hashing function but it should be adequate for moderately effective security.

The following table presents a synopsis of the ServerSocket methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerSocket(void)</td>
<td>Construct a closed server socket</td>
</tr>
<tr>
<td>ServerSocket(int port)</td>
<td>Construct a server socket open on a specified port.</td>
</tr>
<tr>
<td>void open(int port)</td>
<td>Open a closed socket on a specified port</td>
</tr>
<tr>
<td>Socket* accept(void)</td>
<td>Accept a connection on a server socket</td>
</tr>
</tbody>
</table>
The following table presents a synopsis of the ClientSocket methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientSocket(void)</td>
<td>Construct a closed client socket</td>
</tr>
<tr>
<td>ClientSocket(string inet_address, int port)</td>
<td>Construct a client socket and connect to remote server socket</td>
</tr>
<tr>
<td>void connect(string inet_address, int port)</td>
<td>Connect a closed client socket to a specified remote server socket</td>
</tr>
<tr>
<td>Socket* accept(void)</td>
<td>Accept a connection on a server socket</td>
</tr>
</tbody>
</table>

The following table presents a synopsis of the methods available to both client and accept sockets:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int readBuffer(void* msg, int msg_siz)</td>
<td>Read a buffer from socket, returns actual bytes read</td>
</tr>
<tr>
<td>int writeBuffer(void* msg, int msg_siz)</td>
<td>Write a buffer to socket, returns actual bytes written</td>
</tr>
<tr>
<td>unsigned char readByte(void)</td>
<td>Shorthand read routines that deal with proper big-endian conversion</td>
</tr>
<tr>
<td>short readShort(void)</td>
<td></td>
</tr>
<tr>
<td>int readInteger(void)</td>
<td></td>
</tr>
<tr>
<td>long long int readLong(void)</td>
<td></td>
</tr>
<tr>
<td>void readString(string &amp;data)</td>
<td></td>
</tr>
<tr>
<td>void writeByte(unsigned char data)</td>
<td>Shorthand write routines that deal with proper big-endian conversion</td>
</tr>
<tr>
<td>void writeShort(short data)</td>
<td></td>
</tr>
<tr>
<td>void writeInteger(int data)</td>
<td></td>
</tr>
<tr>
<td>void writeLong(long long int data)</td>
<td></td>
</tr>
<tr>
<td>void writeString(string &amp;data)</td>
<td></td>
</tr>
<tr>
<td>bool secureCheck(string password)</td>
<td>Security check on socket</td>
</tr>
<tr>
<td>SocketStatus getStatus(void)</td>
<td>Get status of socket</td>
</tr>
<tr>
<td>void close(void)</td>
<td>Close the socket</td>
</tr>
<tr>
<td>~Socket()</td>
<td>Destroy socket (called via delete), closes first if socket open</td>
</tr>
</tbody>
</table>

No attempt is made at the current time to implement asynchronous connections on these sockets. All calls wait until operation has completed or error occurs.

No attempt is made to handle SIG_PIPE signals, so a broken connection will generally result in a process exit unless handled externally.

### 4.3.2.3.5 Strings

This source contains a set of classes that can be used as a substitute for the C++ Standard Template Library (STL) string package. This version does not use class templates like the STL version does. Because of this there are two significant differences in this package and the STL implementation.

1) This package provides no support for iterators
2) This package does provide overloaded constructors and operators for converting numeric types (other than character) to string.

Also this class is defined with a class name of 'String' to differentiate it from the STL class 'string'. However, to facilitate a quick change to this class from the other class, there is a typedef equivalent named 'string'. This typedef may be commented out if both types of strings are needed in the same application.

No attempt is being made in this header to define in detail what each of the methods does. Generally, the methods follow the conventions of the STL version – please find the documentation for that.

A special note for the overloaded integer constructors and operators. These are somewhat ambiguous with the similar methods for the char type. So, there is no overloaded integer operator for a character, in other words the following work properly:

```cpp
String s('a');    // s becomes "a"
String s(5);      // s becomes "5"
```

But the following might be a bit confusing:

```cpp
char c = 47;
String s(c);      // s is NOT "47" !!!
int n = c;
s = n;           // s is now "47"
```

Finally, this package generally accepts all values for arguments and makes a best attempt to make sense out of them. For example:

```cpp
String s = "ABCD";
s.replace(2,"0123");  // s = "AB0123"
s.insert(100,"0123");   // s becomes "AB01230123"
```

In other words, positions and lengths are modified to reflect what most closely resembles a legal operation.

The following list presents a synopsis of the methods in this class. Note that in the tables, the following symbology conventions apply:

- **s**: a string
- **cs**: C string (pointer to null terminated string)
- **c**: a character
- **n**: a integer number (other than char)
- **f**: a floating number (float or double)
- **is**: istream
- **os**: ostream
- **[x]**: optional
- **opos,olen**: position and length in string object
- **spos,slen**: position and length in string argument
- **cpos,clen**: position and length of C string argument
String();
String(s);
String(s[, spos[, slen]]);
String(cs[, clen]);
String(char c);
String(n, c);
String(n);
String(f);
~String(void);
char at(index);
String &append(c);
String &append(cs[, clen]);
String &append(s[, spos[, slen]]);
String &assign(c);
String &assign(cs[, clen]);
String &assign(s[, spos[, slen]]);
size_type capacity(void) const;
int compare(opos, olen, s[, spos[, slen]]);
int compare(opos, olen, cs[, clen]);
int compare(cs[, clen]);
int compare(s[, spos[, slen]]);
int copy(cs[, olen, opos]) const;
char* c_str();
char* data();
bool empty();
String &erase([opos[, olen]]);
size_type find(c[, opos]);
size_type find(cs[, opos[, clen]]);
size_type find(s[, opos[, slen]]);
size_type find_first_of(c[, opos]);
size_type find_first_of(cs[, opos[, clen]]);
size_type find_first_of(s[, spos[slen]]);
size_type find_first_not_of(c[, opos]);
size_type find_first_not_of(cs[, opos[, clen]]);
size_type find_first_not_of(s[, opos[, slen]]);
size_type find_last_of(c[, opos]);
size_type find_last_of(cs[, opos[, clen]]);
size_type find_last_of(s[, opos[, slen]]);
size_type find_last_not_of(cs[, opos]);
size_type find_last_not_of(cs[, opos[, clen]]);
size_type find_last_not_of(s[, opos[, slen]]);
istream &getline(is, s[, C]);
String &insert(opos, c);
String &insert(opos, cs[, clen]);
String &insert(opos, s[, spos[, slen]]);
String &insert(opos, n, c);
size_type length();
String lower();
size_type max_size();
const String &replace(opos, c);
const String &replace(opos, cs[, clen]);
const String &replace(opos, s[, spos[, slen]]);
const String &replace(opos, n, c);
void resize(n[, c]);
size_type rfind(c[, opos]);
size_type rfind(cs[, opos[, clen]]);
size_type rfind(s[, opos[, slen]]);
size_type size();
String substr([opos[, olen]]);
void swap(s);
String upper();
s = s1;
c = s[n];
s[n] = c;
s += s;
s += cs;
s += c;
s += n;
s += f;
(String) s + s;
(String) cs + s;
(String) s + cs;
(String) cs + s;
(String) s + cs;
(String) n + s;
(String) s + n;
(String) f + s;
(String) s + f;
(bool) s == s;
(bool) cs == s;
(bool) s == cs;
(bool) s != s;
(bool) cs != s;
(bool) s != cs;
(bool) s > s;
(bool) cs > s;
(bool) s > cs;
(bool) s < s;
(bool) cs < s;
(bool) s < cs;
(bool) s >= s;
(bool) cs >= s;
(bool) s >= cs;
(bool) s <= s;
(bool) cs <= s;
(bool) s <= cs;
(ostream) os << s;
(istream) is >> s;
## 4.3.2.3.6 Containers

This source contains a set of classes that can be used to maintain collections of pointers to items. The following are the main classes that make up this package:

- **Container**: an abstract container object
- **Sequence**: an abstract container whose members are stored in order
- **List**: a sequence container that is a dual linked list
- **Vector**: a sequence container that is a dynamically sized vector
- **Map**: an abstract map container
- **HashMap**: a map container that uses hashing (NOTE: UNTESTED !!!)

The Container, Sequence, and Map classes are all abstract and may only be used as a pointer (or reference) to one of the other subclasses. To use the subclasses, create one of the container objects:

```c
Vector defvec();  // Creates a vector with default sizes (64, 64)
Vector vec(128, 256); // Starts with 128 entries, extends 256 at a time
List() alist;  // Creates a linked list
HashMap map(); // Create a map default 512 entries, 25% extend
HashMap map(100, 10.0); // Create a map with 100 entries, 10% extend
```

These classes all store void* pointers so the actual type of Object or variable being referenced is not known to these classes. You can pretty much use any pointer type where an Right Hand Side value is required. Here are a few examples, using map, vec, and alist as defined above:

```c
MyStruct *ms = new MyStruct;
map.put(14, ms);

MyIntegers *mi = new Integer[4654];
vec[14] = mi;

MyClass *mc = new MyClass();
alist.add(mc);
```

However, when reading out of the containers, you will normally have to typecast the results of the method call back to your type. Here are some related examples:

```c
ms = (MyStruct*)map.get(14);
mi = (MyIntegers*)v[14];
mc = (MyClass*)l.pop_back();
```

### Sequence Containers

The Sequence containers are used to maintain collections of pointers that are in some order. These containers actually behave mostly like dynamically sized arrays. There

9/10/09  -  182 -  DCS 104.4
are two types, Vector and List. Which one to use is really dependent on how you want to access the list:

<table>
<thead>
<tr>
<th>Comparison Category</th>
<th>Vector</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Usage</td>
<td>One pointer per item</td>
<td>Two pointers per item</td>
</tr>
<tr>
<td>Random Access</td>
<td>Faster</td>
<td>Slower</td>
</tr>
<tr>
<td>Insert/Delete</td>
<td>Slow except a back</td>
<td>Fast everywhere</td>
</tr>
<tr>
<td>Push/Pop</td>
<td>Slow in front, fast in back</td>
<td>Fast for both</td>
</tr>
</tbody>
</table>

A Vector is probably better for most lists of items where you will not be inserting/removing items into the list other than at the end. Indexing operations into vectors are very fast when compared to a List, particularly if the indexing is not done in order.

However, the List class does do some optimizations to improve performance. Consider a loop such as the following:

```cpp
for (id_type n = 0; n < my_list.count(); n++)
{
    my_item = (MyItem*)my_list[n];
    // Do something with my_item...
}
```

The List class will save its last position so that it knows how far to move to the next item. In this loop, the list class will only have to traverse one link to get to the next entry. So, while random access will not be very efficient, iterating over the list offers acceptable performance.

The index value for all Sequence container methods that accept one is zero relative. It can refer to any position in the list, or in some cases the position immediately past the last item of the list. In general, a reference to an illegal index will cause the container class to throw an exception with a descriptive char* message. Inserting an element into a container causes it to be placed BEFORE the specified index.

If you know that you are going to be using a very large vector, it is strongly suggested that you make the initial and extend sizes adequately large in the constructor for the object. A significant performance penalty is incurred for extending the list.

**Map Containers**

**NOTE - The Map containers are coded, but mostly untested 07/19/2002 ...RMS**

The Map classes are for situations where you want to look up an object by an integer 'key' value. Unlike the Sequence classes, removing or inserting items from or to a map has no effect on the numbering of the remaining objects.
Maps are generally faster at random access than Lists, but slower than Vectors. For a strictly sequential list, they use more memory than either of the other two, but use less when there are gaps in the numbering.

The Map containers do not use indices. Instead, they use unsigned integer 'keys'. No two entries in a map may contain the same key. Any key value may be used at any time without impacting overall container performance. In other words, statistically the following two pairs of instructions should take about the same amount of time - and will use similar amounts of memory:

```
map.put(1,item1); // put 1 with key = 1
map.put(2,item2); // put 2 with key = 2
map.put(1,item1); // put 1 with key = 1
map.put(1000000,item2); // put 2 with key = 1000000
```

As with the Vector class, if you know you are going to use a very large HashMap, it is best to allocate a large initial size for the map. If the container runs out of room, it must resize the hash table, and then rehash all of the values in the table. This can be quite time consuming on large tables, much worse than that of a Vector. Also, use caution when selecting an extend size. The extend size is expressed as a percentage amount by which to expand the list each time a rehash is required. The largest legal value is 200%, which causes the table to triple in size each time it is expanded.

**Limitations**

One **HUGE** consideration is that the `clear` method does not make any attempt to delete the items in the container. You must delete all the items before doing a clear to really clean up.

The objects of this package are VERY DEFINITELY NOT thread safe.

**Synopsis**

The following table presents a synopsis of the methods available in all of these classes:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool isEmpty() const;</code></td>
<td>Returns true if container is empty</td>
</tr>
<tr>
<td><code>size_type size(void) const;</code></td>
<td>Returns number of items in container</td>
</tr>
<tr>
<td><code>void clear(void);</code></td>
<td>Emptys container BUT DOES NOT DELETE ITEMS!!!</td>
</tr>
<tr>
<td><code>bool contains(void *item) const;</code></td>
<td>Determines in container holds item</td>
</tr>
<tr>
<td><code>bool remove(void *item);</code></td>
<td>Removes item from container</td>
</tr>
</tbody>
</table>

The following table presents a synopsis of the methods available in all Sequence classes (List and Vector):
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void add(void *item);</td>
<td>Add item to container at end (same as push_back)</td>
</tr>
<tr>
<td>void add(Sequence &amp;sequence, index_type sindex = 0, size_type ssize = no_size);</td>
<td>Add all or specified items from another to this sequence</td>
</tr>
<tr>
<td>void extract(index_type index);</td>
<td>Remove item from sequence at position and return pointer to it</td>
</tr>
<tr>
<td>void pop_front(void);</td>
<td>Remove the first item from the sequence and return pointer to it</td>
</tr>
<tr>
<td>void pop_back(void);</td>
<td>Remove the last item from the sequence and return pointer to it</td>
</tr>
<tr>
<td>void push_front(void *item);</td>
<td>Add an item to the front of the sequence</td>
</tr>
<tr>
<td>void push_back(void *item);</td>
<td>Add an item to the back of the sequence</td>
</tr>
<tr>
<td>void erase(index_type index, size_type size = no_size);</td>
<td>Remove a range of items from the sequence, does not remove items from memory</td>
</tr>
<tr>
<td>void* get(index_type index);</td>
<td>Get an item at a position in the sequence</td>
</tr>
<tr>
<td>void insert(index_type index, void *item);</td>
<td>Insert an item into the sequence at a position</td>
</tr>
<tr>
<td>void insert(index_type index, Sequence &amp;sequence, index_type sindex = 0, size_type ssize = no_size);</td>
<td>Insert all or specified items from another to this sequence</td>
</tr>
<tr>
<td>index_type indexOf(void *item) const;</td>
<td>Returns the position of an item in the sequence</td>
</tr>
<tr>
<td>void* put(index_type index, void *item);</td>
<td>Replaces item at position in list with new item. Old item is not removed from memory</td>
</tr>
<tr>
<td>void* &amp;operator[](index_type index);</td>
<td>Returns item at position in list</td>
</tr>
</tbody>
</table>

The following table presents a list of the methods unique to the Vector class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector(void);</td>
<td>Construct an empty vector with default sizes</td>
</tr>
<tr>
<td>Vector(size_type isize, size_type esize);</td>
<td>Construct an empty vector with explicit initial and extend sizes</td>
</tr>
<tr>
<td>Vector(Sequence &amp;sequence, index_type index = 0, index_type size = no_size);</td>
<td>Construct a vector from some or all of the items in another sequence</td>
</tr>
<tr>
<td>~Vector(void);</td>
<td>Remove the vector from memory, does not delete items in vector</td>
</tr>
</tbody>
</table>

The following table presents a list of the methods unique to the List class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List(void);</td>
<td>Construct an empty list with default sizes</td>
</tr>
<tr>
<td>List(Sequence &amp;sequence, index_type index = 0, index_type size = no_size);</td>
<td>Construct a list from some or all of the items in another sequence</td>
</tr>
<tr>
<td>~List(void);</td>
<td>Remove the list from memory, does not delete items in list</td>
</tr>
</tbody>
</table>

The following table presents a list of the methods available in all Map classes, currently only HashMap:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool erase(key_type key);</td>
<td>Remove item with specified key</td>
</tr>
<tr>
<td>bool containsKey(const key_type &amp;key) const;</td>
<td>Returns true if map contains item with a key</td>
</tr>
</tbody>
</table>
### void* extract(const key_type &key);

Removes item from list with specified key.

### void* get(const key_type &key) const;

Gets item from list with specified key.

### void* put(const key_type &key, void* item) const;

Replaces item in list with specified key, returns previous item with key if any, or null otherwise.

### The following table presents a list of the methods unique to the List class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashMap(void);</td>
<td>Construct an empty hash map with default size and extend percentage.</td>
</tr>
<tr>
<td>HashMap(index_type isize, float extend_percent = 25.0);</td>
<td>Construct a hashmap with specific initial size and extend percentage.</td>
</tr>
<tr>
<td>~HashMap(void);</td>
<td>Remove the hashmap from memory, does not delete items in list.</td>
</tr>
</tbody>
</table>

#### 4.3.2.3.7 Time

The Time class provides several useful features; time is stored at the nano-second precision, and allows full operator manipulation of time objects. The Time class objects are of two types “relative” and “absolute”. Relative times are quantities of time between events that cannot be expressed as a date time, conversely absolute times can be expressed as a date time.

String conversion logic of the ostream operator understands the difference between absolute times which are printed in unix standard “Mon Apr 30 10:26:51 2001” mode and relative times which are printed based on their size such as “3 days, 4503 seconds” or “85.024 seconds” or “88324 nanoseconds”

For example the following is a use of relative and absolute Time objects:

```cpp
Time t1;
{...some operation to measure...}
Time t2; // t1 and t2 are absolute times
Time t3 = (t2-t1); // t3 is now a relative time
cout << "Operation took " << t3 << endl;
t2 += 100*Seconds_in_a_day;
cout << "The meat in the freezer is good until " << t2 << endl;
```

The public member functions are:
The static printNow() member returns the current local zone absolute time as a string without the need to create an object, such that you can say:

```cpp
cout << "Starting at " << Time::printNow() << endl;
```

You may use the static printZulu() member to return the current GMT absolute time.

The friend function toString(Time) converts a time to a printable string, this method is used internally to support the ostream << operator.

Two friend sleep routines that accept a Time argument are sleepFor( Time ) which allows stored relative times to be slept through, and sleepUntil( Time ) allows stored absolute times to be slept until reached. Both return 0 on success or 1 if awoken early.

The zulu() and local() member functions are provided such that control over absolute time conversions in either local or GMT time zones can be adjusted. Default zone is local.

An update now() member is included to allow a previously instantiated object to be reset to the current time:
The reset(unsigned int seconds, unsigned int nanoseconds) member allows a previously instantiated object to be reset to a specified time, similar to the assignment "=" operator, except allows seconds and nanoseconds to be specified independently.

All additions and subtraction operators are in units of Time objects or int/float seconds.

Adding and subtracting "int"s MAY result in a loss of the nanosecond accuracy. Use "double"s or Time objects to maintain sub-second accuracy.

The following members are for Absolute times only!! Will return –1 when called with relative times.
- julianDay() – this member returns the day-of-year where Jan 1 is integer 1
- year() – this member returns the integer year.
- yyyyymmddhhmmsst() – this member returns a string in the member name form.

4.3.2.3.8 Timer
This source contains an abstract timer class. Users derive subclasses of this class, overriding the onUpdate method. The overridden update method will be called when the timer elapses.

Timers are implemented using a single thread (a descendent of the an abstract Thread) that calls all of the update entry points. This thread is started when the first thread is added, and stops when the last timer is deleted.

Timers allow specification of their update interval in milliseconds. However, the thread sleeps at a period of 10 milliseconds, so timers will usually not be called any more often than that. Furthermore, timers all share the same thread so a long-running onUpdate() method could cause a delay in other methods.

Timers can either be one-shot or repeating. This behavior is controlled by the return of the onUpdate method. If the onUpdate method returns true, the timer will be reenabled for another interval.

It is critical to understand that the onUpdate method is running in a separate thread. This means that you must use Gate objects of the Thread class to synchronize access to complex data structures.

To create a timer one should create a subclass of the Timer class and override the onUpdate method as follows:
// class MyTimer: public Timer
{
    virtual bool onUpdate(void);
}
//
bool MyTimer::onUpdate(void)
{   ...code executed by timer goes here, return true to run repeatedly
}

Then, an instance of a timer object should be created, its interval should be specified, and it should be enabled

//
MyTimer myTimer;               Timer object
//
int main()
{
    myTimer.setInterval(100);   Update every 1/10 second
    myTimer.enable();           Enable the timer
    ...
}

The following table presents a synopsis of the Timer methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer(void);</td>
<td>Construct a timer object, will be disabled with an interval of zero. Note that a Timer can only be constructed through a subclass of Timer, because the OnUpdate method is abstract.</td>
</tr>
<tr>
<td>int setInterval(int milliseconds)</td>
<td>Set the timers interval in milliseconds</td>
</tr>
<tr>
<td>void reset(void)</td>
<td>Reset (disables) the timer. Next update will be set to current time plus the interval.</td>
</tr>
<tr>
<td>void enable(void)</td>
<td>Enables timer</td>
</tr>
<tr>
<td>virtual bool onUpdate(void)</td>
<td>Abstract virtual method, must be overridden. Methods will be called at approximately update interval after enable, and periodically thereafter if true is returned. OnUpdate methods must not take inordinate amounts of time and should never sleep because all OnUpdate methods run from the same thread.</td>
</tr>
<tr>
<td>~Timer(void);</td>
<td>Remove the Timer from memory, may also remove thread if last timer object</td>
</tr>
</tbody>
</table>

4.3.2.3.9 Strutils
The StrUtils source contains a set of functions that can be used to convert numeric arguments to a string. The following is a list of the available calls:
4.4 Moving Window Display Component

The MWD Component is responsible for accepting commands from the MWDFormatter. The MWD component will be basically re-used “as-is”.

```cpp
string toString(long long int value);
string toString(long long int value, int precision);
string toString(long long int value, int width, int precision);
string toString(int value);
string toString(int value, int precision);
string toString(int value, int width, int precision);
string toString(short int value);
string toString(short int value, int precision);
string toString(short int value, int width, int precision);
string toString(unsigned char value);
string toString(unsigned char value, int precision);
string toString(unsigned char value, int width, int precision);
string toString(signed char value);
string toString(signed char value, int precision);
string toString(signed char value, int width, int precision);
string toString(double value);
string toString(double value, int precision);
string toString(double value, int width, int precision);
string toString(float value);
string toString(float value, int precision);
string toString(float value, int width, int precision);
```
## Section 5  Requirements Traceability

<table>
<thead>
<tr>
<th>SRS</th>
<th>SDD Software Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.1.1</td>
<td>n/a</td>
</tr>
</tbody>
</table>
| 3.1.1.1.1 | 3.1.1.2.3 mac_GenSuptSchedule  
            3.1.1.2.5 mac_SendSetup  
            3.1.1.2.6 mac_ManCapture |
| 3.1.1.1.2 | 3.1.1.2.3 mac_GenSuptSchedule  
            3.1.1.2.5 mac_SendSetup  
            3.1.1.2.6 mac_ManCapture |
| 3.1.1.2 | n/a |
| 3.1.1.2.1 | n/a |
| 3.1.1.2.1.1 | 3.1.1.2.5 mac_SendSetup |
| 3.1.1.2.1.2 | 3.1.1.2.5 mac_SendSetup |
| 3.1.1.2.1.3 | 3.1.1.2.5 mac_SendSetup |
| 3.1.1.2.1.4 | 3.1.1.2.5 mac_SendSetup |
| 3.1.1.2.2 | 3.1.1.7.2 mac_ui_edit_capt_parms  
            3.1.1.2.5 mac_SendSetup |
| 3.1.1.2.3 | 3.1.1.2.5 mac_SendSetup |
| 3.1.1.3 | n/a |
| 3.1.1.3.1 | 3.1.1.2.6 mac_ManCapture |
| 3.1.1.3.1.1 | 3.1.1.2.6 mac_ManCapture |
| 3.1.1.3.1.2 | 3.1.1.2.6 mac_ManCapture |
| 3.1.1.3.1.3 | 3.1.1.2.6 mac_ManCapture |
| 3.1.1.3.1.4 | 3.1.1.7.6 mac_ui_manual_capt  
            3.1.1.2.6 mac_ManCapture |
| 3.1.1.3.2 | 3.1.1.7.6 mac_ui_manual_capt  
            3.1.1.2.6 mac_ManCapture |
| 3.1.1.3.3 | 3.1.1.2.6 mac_ManCapture |
| 3.1.1.3.4 | 3.1.1.2.6 mac_ManCapture |
| 3.1.1.4 | n/a |
| 3.1.1.4.1 | 3.1.1.2.3 mac_GenSuptSchedule  
            3.1.1.2.3 mac_GenSuptSchedule |
| 3.1.1.4.2 | 3.1.1.7.3 mac_ui_edit_cont_sch  
            3.1.1.2.3 mac_GenSuptSchedule |
| 3.1.1.4.3 | 3.1.1.2.3 mac_GenSuptSchedule  
            3.1.1.2.3 mac_GenSuptSchedule |
| 3.1.1.4.4 | 3.1.1.2.3 mac_GenSuptSchedule  
            3.1.1.2.3 mac_GenSuptSchedule |
| 3.1.1.4.5 | 3.1.1.2.3 mac_GenSuptSchedule  
            3.1.1.2.3 mac_GenSuptSchedule |
<p>| 3.1.1.5 | n/a |
| 3.1.1.5.1 | 3.1.2.6 Raw File Transfer Component (CTS) |
| 3.1.1.5.2 | 3.1.2.1 AutoCapture Component |
| 3.1.1.5.3 | 3.1.2.6 Raw File Transfer Component (CTS) |
| 3.1.1.5.4 | 3.1.2.2 Raw Data Capture Component |
| 3.1.1.5.4.1 | 3.1.2.2 Raw Data Capture Component |
| 3.1.1.5.4.2 | 3.1.2.2 Raw Data Capture Component |
| 3.1.1.5.4.3 | 3.1.2.2 Raw Data Capture Component |
| 3.1.1.5.4.4 | 3.1.2.2 Raw Data Capture Component |
| 3.1.1.5.4.5 | 3.1.2.2 Raw Data Capture Component |</p>
<table>
<thead>
<tr>
<th>SRS</th>
<th>SDD Software Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1.5.4.6</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.4.7</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.4.8</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.5</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.2</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.3</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.4</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.5</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.6</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.7</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.8</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.9</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.10</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.11</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.12</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.13</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.14</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.15</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.16</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.1.5.6.17</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.1.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.2.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.2.1.1</td>
<td>3.1.1.1.1 Table MISSION_ACCT</td>
</tr>
<tr>
<td>3.1.2.1.2</td>
<td>3.1.1.1.1 Table MISSION_ACCT</td>
</tr>
<tr>
<td>3.1.2.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.2.2.1</td>
<td>3.1.1.1.2 Table DESTINATION_ACCT</td>
</tr>
<tr>
<td>3.1.2.2.2</td>
<td>3.1.1.1.2 Table DESTINATION_ACCT</td>
</tr>
<tr>
<td>3.1.2.2.3</td>
<td>3.1.1.1.2 Table DESTINATION_ACCT</td>
</tr>
<tr>
<td>3.1.2.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.2.3.1</td>
<td>3.1.1.1.3 Table ROUTING_ACCT</td>
</tr>
<tr>
<td>3.1.2.3.2</td>
<td>3.1.1.1.3 Table ROUTING_ACCT</td>
</tr>
<tr>
<td>3.1.2.4</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.2.4.1</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT</td>
</tr>
<tr>
<td>3.1.2.4.2</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT</td>
</tr>
<tr>
<td>3.1.2.4.3</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT</td>
</tr>
<tr>
<td>3.1.2.4.4</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT</td>
</tr>
<tr>
<td>3.1.2.4.5</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT</td>
</tr>
<tr>
<td>3.1.2.5</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.2.5.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.2.5.2</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT</td>
</tr>
<tr>
<td>3.1.2.5.3</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT</td>
</tr>
<tr>
<td>3.1.3</td>
<td>n/a</td>
</tr>
<tr>
<td>SRS</td>
<td>SDD Software Unit</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>3.1.3.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.3.1.1</td>
<td>3.1.1.2.2 mac_IngestContact</td>
</tr>
<tr>
<td>3.1.3.1.2</td>
<td>3.1.1.2.2 mac_IngestContact</td>
</tr>
<tr>
<td>3.1.3.1.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.1.3.1.4</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.1.1</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.1.2</td>
<td>3.1.1.2.2 mac_IngestContact</td>
</tr>
<tr>
<td>3.2.1.1.3</td>
<td>3.1.1.2.2 mac_IngestContact</td>
</tr>
<tr>
<td>3.2.1.1.4</td>
<td>3.1.1.2.2 mac_IngestContact</td>
</tr>
<tr>
<td>3.2.1.1.5</td>
<td>3.1.1.2.3 mac_GenSuptSchedule</td>
</tr>
<tr>
<td>3.2.1.1.6</td>
<td>3.1.1.2.6 mac_ManCapture</td>
</tr>
<tr>
<td>3.2.1.1.7</td>
<td>3.1.1.2.5 mac_SendSetup</td>
</tr>
<tr>
<td>3.2.1.1.8</td>
<td>3.1.1.2.6 mac_ManCapture</td>
</tr>
<tr>
<td>3.2.1.1.9</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.1.10</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.1.11</td>
<td>3.1.1.7.9 mac_ui_DDS_config</td>
</tr>
<tr>
<td>3.2.1.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.2.1</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.2</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.2.2.1</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.2.2.2</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.2.2.3</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.2.3</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.4</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.5</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.6</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.7</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.7.1</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.7.2</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.7.3</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.7.4</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.8</td>
<td>3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.2.9</td>
<td>3.1.1.1.12 Table DCS_CONFIGURATION,</td>
</tr>
<tr>
<td>3.2.1.2.10</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.1</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.2</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.2.1</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.2.2</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.2.3</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.3</td>
<td>3.1.1.3.1 mac_BackupArchive</td>
</tr>
<tr>
<td>SRS</td>
<td>SDD Software Unit</td>
</tr>
<tr>
<td>-----</td>
<td>-------------------</td>
</tr>
<tr>
<td>3.2.1.2.10.3.1</td>
<td>3.1.1.3.1 mac BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.3.2</td>
<td>3.1.1.3.1 mac BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.10.3.3</td>
<td>3.1.1.3.1 mac BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.11</td>
<td>3.1.1.3.1 mac BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.11.1</td>
<td>3.1.1.3.1 mac BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.11.2</td>
<td>3.1.1.3.1 mac BackupArchive</td>
</tr>
<tr>
<td>3.2.1.2.12</td>
<td>3.1.1.3.1 mac BackupArchive</td>
</tr>
<tr>
<td>3.2.1.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.3.1</td>
<td>3.1.1.5.1 mac Restage</td>
</tr>
<tr>
<td>3.2.1.3.2</td>
<td>3.1.1.5.1 mac Restage</td>
</tr>
<tr>
<td>3.2.1.3.3</td>
<td>3.1.1.5.1 mac Restage</td>
</tr>
<tr>
<td>3.2.1.3.4</td>
<td>3.1.1.5.1 mac Restage</td>
</tr>
<tr>
<td>3.2.1.3.5</td>
<td>3.1.1.5.1 mac Restage</td>
</tr>
<tr>
<td>3.2.1.4</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.4.1</td>
<td>3.1.1.4.1 mac DeleteRawFiles</td>
</tr>
<tr>
<td>3.2.1.4.1.1</td>
<td>3.1.1.4.1 mac DeleteRawFiles</td>
</tr>
<tr>
<td>3.2.1.4.1.2</td>
<td>3.1.1.4.1 mac DeleteRawFiles</td>
</tr>
<tr>
<td>3.2.1.4.1.3</td>
<td>3.1.1.4.1 mac DeleteRawFiles</td>
</tr>
<tr>
<td>3.2.1.4.2</td>
<td>3.1.1.4.1 mac DeleteRawFiles</td>
</tr>
<tr>
<td>3.2.1.4.3</td>
<td>3.1.1.4.1 mac DeleteRawFiles</td>
</tr>
<tr>
<td>3.2.1.5</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.5.1</td>
<td>3.1.1.1.2 Table DESTINATION_ACCT, 3.1.1.1.3 Table ROUTING_ACCT, 3.1.1.1.9 Table TRANSFER_ACCT, 3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.5.2</td>
<td>3.1.1.1.9 Table TRANSFER_ACCT, 3.1.1.2.1 mac_DDS</td>
</tr>
<tr>
<td>3.2.1.6</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.6.1</td>
<td>3.1.1.7.12 mac_ui_rawfile_details</td>
</tr>
<tr>
<td>3.2.1.6.2</td>
<td>3.1.1.7.1 mac_ui_DDS_main, 3.1.1.7.12 mac_ui_rawfile_details</td>
</tr>
<tr>
<td>3.2.1.6.3</td>
<td>3.1.1.7.10 mac_ui_mission_config</td>
</tr>
<tr>
<td>3.2.1.6.4</td>
<td>3.1.1.7.12 mac_ui_rawfile_details</td>
</tr>
<tr>
<td>3.2.1.6.5</td>
<td>3.1.1.7.3 mac_ui_edit_cont_sched</td>
</tr>
<tr>
<td>3.2.1.6.6</td>
<td>3.1.1.7.3 mac_ui_edit_cont_sched, 3.1.1.7.6 mac_ui_manual_capt, 3.1.1.7.12 mac_ui_rawfile_details</td>
</tr>
<tr>
<td>3.2.1.6.7</td>
<td>3.1.1.7.6 mac_ui_manual_capt</td>
</tr>
<tr>
<td>3.2.1.6.8</td>
<td>3.1.1.7.1 mac_ui_DDS_main</td>
</tr>
<tr>
<td>3.2.1.6.9</td>
<td>3.1.1.7.1 mac_ui_DDS_main</td>
</tr>
<tr>
<td>3.2.1.6.10</td>
<td>3.1.1.7.1 mac_ui_DDS_main</td>
</tr>
<tr>
<td>SRS</td>
<td>SDD Software Unit</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>3.2.1.6.11</td>
<td>3.1.1.7.2 mac_ui_edit_capt_parms,</td>
</tr>
<tr>
<td></td>
<td>3.1.1.7.9 mac_ui_DDS_config,</td>
</tr>
<tr>
<td></td>
<td>3.1.1.7.10 mac_ui_mission_config,</td>
</tr>
<tr>
<td></td>
<td>3.1.1.7.11 mac_ui_dest_config</td>
</tr>
<tr>
<td>3.2.1.6.12</td>
<td>3.1.1.7.2 mac_ui_edit_capt_parms,</td>
</tr>
<tr>
<td></td>
<td>3.1.1.7.9 mac_ui_DDS_config</td>
</tr>
<tr>
<td>3.2.1.7</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.1.7.1</td>
<td>3.1.1.6.1 mac_JournalFileEntry</td>
</tr>
<tr>
<td>3.2.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.2.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.2.1.1</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.2</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.3</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.3.1</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.3.2</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.4</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.5</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.6</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.7</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.8</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.8.1</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.8.2</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.9</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.1</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.2</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.3</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.4</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.5</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.6</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.7</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.10.8</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.11</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.12</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.13</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.13.1</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.13.2</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.13.3</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.14</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.14.1</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.14.2</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.1.14.3</td>
<td>3.1.2.1 AutoCapture Component</td>
</tr>
<tr>
<td>3.2.2.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.2.2.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>SRS</td>
<td>SDD Software Unit</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>3.2.2.1.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.1.2</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.2</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.3</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.4</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.5</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.5.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.5.2</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.5.3</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.5.4</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.5.5</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.6</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.7</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.7.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.7.2</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.7.3</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.7.4</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.8</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.2.9</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.2.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.3.1</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.2</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.3</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.4</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.5</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.6</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.7</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.8</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.8.1</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.8.2</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.8.3</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.8.4</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.9</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.10</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.11</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.12</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.2.3.13</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.4</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.4.1</td>
<td>3.1.2.5 Delete Raw Files Component (CTS)</td>
</tr>
<tr>
<td>3.2.4.2</td>
<td>3.1.2.5 Delete Raw Files Component (CTS)</td>
</tr>
<tr>
<td>3.2.4.3</td>
<td>3.1.2.5 Delete Raw Files Component (CTS)</td>
</tr>
<tr>
<td>3.2.4.4</td>
<td>3.1.2.5 Delete Raw Files Component (CTS)</td>
</tr>
<tr>
<td>3.2.4.5</td>
<td>3.1.2.5 Delete Raw Files Component (CTS)</td>
</tr>
<tr>
<td>3.2.5</td>
<td>n/a</td>
</tr>
<tr>
<td>SRS</td>
<td>SDD Software Unit</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>3.2.2.5.1</td>
<td>3.1.2.6 Raw File Transfer Component (CTS)</td>
</tr>
<tr>
<td>3.2.2.5.2</td>
<td>3.1.2.6 Raw File Transfer Component (CTS)</td>
</tr>
<tr>
<td>3.2.2.5.3</td>
<td>3.1.2.6 Raw File Transfer Component (CTS)</td>
</tr>
<tr>
<td>3.2.2.5.4</td>
<td>3.1.2.6 Raw File Transfer Component (CTS)</td>
</tr>
<tr>
<td>3.2.2.5.5</td>
<td>3.1.2.6 Raw File Transfer Component (CTS)</td>
</tr>
<tr>
<td>3.2.2.5.6</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.2.6</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.2.6.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.2.7</td>
<td>n/a</td>
</tr>
<tr>
<td>3.2.2.7.1</td>
<td>3.1.2.7 Raw Data Transmit Component</td>
</tr>
<tr>
<td>3.3.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.1.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.3.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.4</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.5</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.6</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.7</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.8</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.1.9</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.1.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.3.2.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.2.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.4</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.4.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.3.2.5</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.5.1</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.3.2.5.2</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.3.2.5.3</td>
<td>3.1.2.3 Transfer To Tape Component</td>
</tr>
<tr>
<td>3.3.2.6</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.7</td>
<td>n/a</td>
</tr>
<tr>
<td>3.3.2.7.1</td>
<td>3.1.2.6 Raw File Transfer Component (CTS)</td>
</tr>
<tr>
<td>3.3.2.7.2</td>
<td>3.1.2.6 Raw File Transfer Component (CTS)</td>
</tr>
<tr>
<td>3.5.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.4</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.4.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.4.1.1</td>
<td>n/a</td>
</tr>
<tr>
<td>SRS</td>
<td>SDD Software Unit</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>3.5.4.1.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.4.1.3</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.4.1.4</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.4.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.4.2.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.5</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.5.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.5.1.1</td>
<td>3.1.2.2 Raw Data Capture Component</td>
</tr>
<tr>
<td>3.5.5.1.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.5.2</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.5.2.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.6</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.7</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.8</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.9</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.10</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.10.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.10.1.1</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.11</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.12</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.13</td>
<td>n/a</td>
</tr>
<tr>
<td>3.5.14</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Section 6  Notes

6.1  Raw Data File Queue

The raw data file queue will be first-in/first-out based on the scheduled start time and priority. When accounting files are ingested all information is inserted into table DCS_RAWFILE_ACCT and the ON_LINE_FLAG is set to 1. The SCHEDULED_START_TIME and PRIORITY fields determine the next available file in the queue (this is accomplished by mac_DDS polling function Check_for_xfers). TRANSFER_ACCT records are also created to notify each destination that the file is ready. The TRANSFER_ACCT and DCS_RAWFILE_ACCT define the raw file queues.

Each mission will have multiple logical queues, one for each "logical" processing system and one for each "logical" archive. By "logical", each destination system may have more than one node that can pull raw files from the same queue; however, the file is only retrieved by one of the nodes and once pulled, the file is marked (by that node) as successfully transferred. There is one TRANSFER_ACCT record created for each logical destination.

During the polling, when a file transfer is completed (all destinations have set their XFER_STATUS field to ‘SUCCESS’ and the file is still on-line) and there are no other "READY" files in the queue, the oldest "IN-QUEUE" file in the same logical queue with the highest priority (lowest value) is set to "READY".
Figure 6-1 Raw Data Queue
## Appendix A  APPENDIXES

### A  DDS_OPS_DB Data Dictionary

#### A.1 MISSION_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISSION_ID</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Mission identification.</td>
<td>N/A</td>
<td>Primary key, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>MISSION_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>Name of mission/data (e.g. ‘Landsat 7 Validation’)</td>
<td>''</td>
<td>Not Null, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>RAW_DATA_PATH</td>
<td>VARCHAR2</td>
<td>512</td>
<td>DDS directory for mission/data raw data.</td>
<td>'/rawdata'</td>
<td>Not Null, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>NUMBER</td>
<td>1</td>
<td>Default priority for the mission.</td>
<td>1</td>
<td>Not Null, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR2</td>
<td>3</td>
<td>Data type identification (e.g. ‘VAL’)</td>
<td>‘NOM’</td>
<td>Primary key, privileged, foreign key to DCS_DATATYPE_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>BIT_RATE</td>
<td>NUMBER</td>
<td>11</td>
<td>Nominal data rate in bits per second.</td>
<td>N/A</td>
<td>None</td>
<td>DDS</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>CHAR</td>
<td>1</td>
<td>Capture device channel mask (e.g. ‘a’ for channel A, ‘b’ for B, ‘*’ for both)</td>
<td>''</td>
<td>None</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 1: MISSION_ACCT**

#### A.2 MISSION_STATION_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISSION_ID</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Foreign key for mission identification.</td>
<td>N/A</td>
<td>Primary key, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>STATION_ID</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Identifier of ground station (e.g. ‘AGS’, ‘EDC’</td>
<td>N/A</td>
<td>Primary key, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR2</td>
<td>3</td>
<td>Data type identification (e.g. ‘VAL’)</td>
<td>‘NOM’</td>
<td>Primary key, privileged, foreign key to DCS_DATATYPE_ACCT</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 2: MISSION_STATION_ACCT**

#### A.3 DESTINATION_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEST_SYS_ID</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Primary key for destination system identification (e.g. ‘LPS-NG’)</td>
<td>N/A</td>
<td>Primary key, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>DEST_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>Name of destination (e.g. ‘Landsat 7 Processing System – Next Generation’)</td>
<td>''</td>
<td>Not Null, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>ARCHIVE_FLAG</td>
<td>NUMBER</td>
<td>1</td>
<td>1 = system permanently archives data</td>
<td>0</td>
<td>Not Null, privileged</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 3: DESTINATION_ACCT**
### A.4 ROUTING_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEST_SYS_ID</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Foreign key for destination system identification.</td>
<td>N/A</td>
<td>Primary key, Foreign key to DESTINATION_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>MISSION_ID</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Foreign key for mission identification.</td>
<td>N/A</td>
<td>Primary key, Foreign key to MISSION_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>INITIAL_XFER_STATUS</td>
<td>VARCHAR2</td>
<td>12</td>
<td>Initial value to use in when creating the TRANSFER_ACCT.XFER_STATUS (e.g., 'IN-QUEUE' for immediate queuing or 'HOLD-XXX' to hold for some external action).</td>
<td>N/A</td>
<td>Not null. DDS</td>
<td>DDS</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR2</td>
<td>3</td>
<td>Data type identification (e.g., 'VAL')</td>
<td>'NOM'</td>
<td>Primary key, privileged, foreign key to DCS_DATATYPE_ACCT</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 4: ROUTING_ACCT**

### A.5 CONTACT_SCHEDULES

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEDULED_START_TIME</td>
<td>DATE</td>
<td>-</td>
<td>Start time of scheduled live data capture.</td>
<td>N/A</td>
<td>Not null. DDS</td>
<td>DDS</td>
</tr>
<tr>
<td>SCHEDULED_STOP_TIME</td>
<td>DATE</td>
<td>-</td>
<td>End time of scheduled live data capture.</td>
<td>N/A</td>
<td>Not null. DDS</td>
<td>DDS</td>
</tr>
<tr>
<td>MISSION_ID</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Foreign key for mission identification.</td>
<td>N/A</td>
<td>Foreign key to MISSION_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>NUMBER</td>
<td>1</td>
<td>Priority (data with greater priority values are to be processed/archived ahead of lesser priority values).</td>
<td>1</td>
<td>0 to 9, not null. DDS</td>
<td>DDS</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR2</td>
<td>3</td>
<td>Data type identification (e.g., 'VAL')</td>
<td>'NOM'</td>
<td>Primary key, privileged, foreign key to DCS_DATATYPE_ACCT</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 5: CONTACT_SCHEDULES**
## A.6 DCS_CAPTURE_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPT_SYS_ID</td>
<td>VARCHAR2</td>
<td>20</td>
<td>Primary key for capture system identification.</td>
<td>N/A</td>
<td>Primary key, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>SOFTWARE_VER_NUM</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Version of CTS software.</td>
<td>N/A</td>
<td>not null</td>
<td>DDS</td>
</tr>
<tr>
<td>ISOLATE_PROCESS</td>
<td>NUMBER</td>
<td>1</td>
<td>1=capture process is to run on an isolated processor, 0=run on shared processor</td>
<td>1</td>
<td>0 or 1, not null</td>
<td>DDS</td>
</tr>
<tr>
<td>SUSPEND_PROCESS</td>
<td>NUMBER</td>
<td>1</td>
<td>1=suspend non-essential processes which may interfere with capture, 0=don’t</td>
<td>1</td>
<td>0 or 1, not null</td>
<td>DDS</td>
</tr>
<tr>
<td>IP_ADDRESS</td>
<td>VARCHAR2</td>
<td>100</td>
<td>Internet protocol address of system for FTP deliveries from the CTS.</td>
<td></td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>User name on capture system (for ftp transfers to the CTS).</td>
<td></td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>VARCHAR2</td>
<td>100</td>
<td>Password on capture system (for ftp transfers to the CTS).</td>
<td></td>
<td>Not null, password</td>
<td>DDS</td>
</tr>
<tr>
<td>SCHEDULE_DIR</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Directory on capture system to place support schedules (for ftp).</td>
<td></td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>PARAMETER_DIR</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Directory on capture system to place capture parameters (for ftp).</td>
<td></td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>TRANSFER_OPTION</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Option for transferring captured files (NONE=transfer disabled, FTP= ftp to DDS, TAPE= copy to tape)</td>
<td>FTP</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>DELETE_RAW_FILE</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Option for deleting raw files after transfer (NO=no delete, YES=delete)</td>
<td>1</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>XFER_IDLE_TIME</td>
<td>NUMBER</td>
<td>5</td>
<td>Amount of time needed before next scheduled capture in order to transfer file (ftp or tar)</td>
<td>0</td>
<td>Not null, 0-99999</td>
<td>DDS</td>
</tr>
<tr>
<td>MWD_NAME</td>
<td>VARCHAR2</td>
<td>30</td>
<td>Name (or number) of MWD system assigned to this CTS.</td>
<td>1</td>
<td>None</td>
<td>DDS</td>
</tr>
<tr>
<td>TAPE_DRIVE</td>
<td>VARCHAR2</td>
<td>50</td>
<td>Device pathname of assigned tape drive.</td>
<td>N/A</td>
<td>None</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 6: DCS_CAPTURE_ACCT**
### A.7 CAPTURE_MISSION_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPT_SYS_ID</td>
<td>VARCHAR2</td>
<td>20</td>
<td>Foreign key for capture system identification.</td>
<td>N/A</td>
<td>Primary key, Foreign key to DCS_CAPTURE_ACCT, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>MISSION_ID</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Foreign key for mission identification.</td>
<td>N/A</td>
<td>Primary key, Foreign key to MISSION_ACCT, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR2</td>
<td>3</td>
<td>Data type identification (e.g. 'VAL')</td>
<td>NOM</td>
<td>Foreign key to DCS_DATATYPE_ACCT</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 7: CAPTURE_MISSION_ACCT**

### A.8 DCS_CONTACT_SCHED_FILES

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_NAME</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Name of contact schedule file</td>
<td>N/A</td>
<td>not null</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 8: DCS_CONTACT_SCHED_FILES**
### TABLE 9: DCS_CONFIGURATION

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCS_HW_STRING_ID</td>
<td>VARCHAR2</td>
<td>30</td>
<td>Name of DDS (i.e. 'DDS')</td>
<td>DDS</td>
<td>not null</td>
<td>DDS</td>
</tr>
<tr>
<td>SOFTWARE_VER_NUM</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Version of DDS software.</td>
<td>N/A</td>
<td>not null</td>
<td>DDS</td>
</tr>
<tr>
<td>SCHEDULE_POLL</td>
<td>NUMBER</td>
<td>3</td>
<td>Number of seconds between checks for new contact schedules, 0=disable</td>
<td>10</td>
<td>not null</td>
<td>DDS</td>
</tr>
<tr>
<td>TRANSFER_POLL</td>
<td>NUMBER</td>
<td>3</td>
<td>Number of seconds between checks for completed transfers to destinations, 0=disable</td>
<td>10</td>
<td>not null</td>
<td>DDS</td>
</tr>
<tr>
<td>DISK_SPACE_POLL</td>
<td>NUMBER</td>
<td>3</td>
<td>Number of seconds between checks for disk space exceeding a threshold.</td>
<td>10</td>
<td>not null</td>
<td>DDS</td>
</tr>
<tr>
<td>DISK_SPACE_THRESH</td>
<td>NUMBER</td>
<td>3</td>
<td>Percent of free disk space remaining to activate automated cleanup.</td>
<td>50</td>
<td>not null, 0-100</td>
<td>DDS</td>
</tr>
<tr>
<td>DELETION_DELAY</td>
<td>NUMBER</td>
<td>6</td>
<td>Number of seconds after last successful transfer to wait before the file is automatically deleted from on-line storage.</td>
<td>0</td>
<td>0-999999</td>
<td>DDS</td>
</tr>
<tr>
<td>FTP_LOGIN_NAME</td>
<td>VARCHAR2</td>
<td>20</td>
<td>FTP login name used by the CTS's when transferring raw files to the DDS.</td>
<td>n/a</td>
<td></td>
<td>DDS</td>
</tr>
<tr>
<td>FTP_PASSWORD</td>
<td>VARCHAR2</td>
<td>20</td>
<td>Password for FTP_LOGIN_NAME</td>
<td>n/a</td>
<td>not null, password</td>
<td>DDS</td>
</tr>
<tr>
<td>FTP_RAWFILE_DIR</td>
<td>VARCHAR2</td>
<td>50</td>
<td>FTP login directory used by the CTS's when transferring raw files to the DDS.</td>
<td>/u01/rawfile</td>
<td>not null</td>
<td>DDS</td>
</tr>
<tr>
<td>TAPE_DRIVE</td>
<td>VARCHAR2</td>
<td>50</td>
<td>Device pathname of assigned tape drive.</td>
<td>N/A</td>
<td>None</td>
<td>DDS</td>
</tr>
</tbody>
</table>

Comment [bjp6]: This should be 512!!
### A.10 DCS_RAWFILE_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPT_SYS_ID</td>
<td>VARCHAR2</td>
<td>20</td>
<td>Foreign key for capture system identification. Indicates which CTS captured the file.</td>
<td>N/A</td>
<td>Foreign key to DCS_CAPTURE_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>CAPTURE_SOURCE</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Ground station channel identification delivering data to capture system (source of capture).</td>
<td>&quot;&quot;</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>SCHEDULED_START_TIME</td>
<td>DATE</td>
<td>n/a</td>
<td>Starting time of downlink from satellite.</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>SCHEDULED_STOP_TIME</td>
<td>DATE</td>
<td>n/a</td>
<td>Ending time of downlink from satellite.</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>ACTUAL_START_TIME</td>
<td>DATE</td>
<td>n/a</td>
<td>Starting time of capture on CTS.</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>ACTUAL_STOP_TIME</td>
<td>DATE</td>
<td>n/a</td>
<td>Ending time of capture on CTS.</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>RAW_DATA_FILE_NAME</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Filename of raw data file (excluding path).</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>RECEIVED_DATA_VOL</td>
<td>NUMBER</td>
<td>7.2</td>
<td>Number of megabytes received (actual number of bytes received, should be same as FILE_SIZE unless the disk fills up)</td>
<td>0</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>EXPECTED_DATA_VOL</td>
<td>NUMBER</td>
<td>7.2</td>
<td>Number of megabytes expected (based on total duration of capture, which is from minimum start time to maximum stop time plus a 'pad').</td>
<td>0</td>
<td>DDS</td>
<td></td>
</tr>
<tr>
<td>SCHEDULED_DATA_VOL</td>
<td>NUMBER</td>
<td>7.2</td>
<td>Number of megabytes scheduled (this should always be the same as 'expected')</td>
<td>0</td>
<td>DDS</td>
<td></td>
</tr>
<tr>
<td>FILE_SIZE</td>
<td>NUMBER</td>
<td></td>
<td>Size of file in bytes.</td>
<td>0</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>TRANSMISSION_RATE</td>
<td>NUMBER</td>
<td>7.2</td>
<td>Number of megabytes per second during capture (calculated from total capture duration).</td>
<td>0</td>
<td>DDS</td>
<td></td>
</tr>
<tr>
<td>ISOLATE_FLAG</td>
<td>NUMBER</td>
<td>1</td>
<td>1=capture process was run on an isolated processor. 0=it wasn’t</td>
<td>0</td>
<td>DDS</td>
<td></td>
</tr>
<tr>
<td>SUSPEND_FLAG</td>
<td>NUMBER</td>
<td>1</td>
<td>1=non-essential processes suspended during capture, 0=they weren’t</td>
<td>0</td>
<td>DDS</td>
<td></td>
</tr>
<tr>
<td>ON_LINE_FLAG</td>
<td>NUMBER</td>
<td>1</td>
<td>1=the raw file is on-line (on disk).</td>
<td>0</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>ORIGIN_STATION</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Ground station that received the data from the satellite.</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>MISSION_ID</td>
<td>VARCHAR2</td>
<td>5</td>
<td>Foreign key for mission identification.</td>
<td>N/A</td>
<td>Foreign key to MISSION_ACCT, privileged</td>
<td>DDS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>NUMBER</td>
<td>1</td>
<td>Priority (data with greater priority values are to be processed/archived ahead of lesser priority values).</td>
<td>N/a 0 to 9, not null</td>
<td>DDS</td>
<td></td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR2</td>
<td>3</td>
<td>Data type identification (e.g. 'VAL')</td>
<td>NOM</td>
<td>Foreign key to DCS_DATATYPE_ACCT</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 10: DCS_RAWFILE_ACCT**

Comment [bjp7]: Need to change to bytes.
### A.11 TRANSFER_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW_DATA_FILE_NAME</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Primary key, foreign key identifies raw data file.</td>
<td>N/A</td>
<td>Primary key, foreign key to DCS_RAWFILE_ACC</td>
<td>DDS</td>
</tr>
<tr>
<td>DEST_SYS_ID</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Primary key, foreign key for destination system identification.</td>
<td>N/A</td>
<td>Primary key, Foreign key to DESTINATION_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>XFER_STATUS</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Status of transfer: READY = data is ready for transfer PROGRESS = transfer in progress SUCCESS = data transferred and verified FAIL-XXX = dest. failure code HOLD-XXX = file is on-hold. IN-QUEUE = file is in queue pending a higher priority or older file (do not xfer).</td>
<td>N/A</td>
<td>Not null, restricted to one of the values defined. Will be initialized by DDS and updated by the destination system defined by DEST_SYS_ID.</td>
<td>DDS</td>
</tr>
<tr>
<td>STATUS_DATE</td>
<td>DATE</td>
<td>n/a</td>
<td>Date/time of last XFER_STATUS update.</td>
<td></td>
<td>System time. Will be initialized by DDS and updated by the destination system defined by DEST_SYS_ID.</td>
<td>DDS / Destination</td>
</tr>
</tbody>
</table>

| TABLE 11: TRANSFER_ACCT |
### A.12 TRANSFER_ACCT_ARCHIVE

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW_DATA_FILE_NAME</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Primary key, foreign key identifies raw data file.</td>
<td>N/A</td>
<td>Primary key, foreign key to DCS_RAWFILE_ACC</td>
<td>DDS</td>
</tr>
<tr>
<td>DEST_SYS_ID</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Primary key, foreign key for destination system identification.</td>
<td>N/A</td>
<td>Primary key, Foreign key to DESTINATION_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>XFER_STATUS</td>
<td>VARCHAR2</td>
<td>10</td>
<td>Status of transfer attempt: SUCCESS = data transferred and verified, FAIL-XXXX = failure defined by dest.</td>
<td>N/A</td>
<td>Not null. Will be copied by DDS from the TRANSFER_ACCT table for a &quot;resend&quot;.</td>
<td>DDS</td>
</tr>
<tr>
<td>STATUS_DATE</td>
<td>DATE</td>
<td>n/a</td>
<td>Date/time of last XFER_STATUS update. system time</td>
<td>not null</td>
<td>not null. Will be copied by DDS.</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 12: TRANSFER_ACCT_ARCHIVE**

### A.13 BACKUP_ACCT

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAPE_ID</td>
<td>VARCHAR2</td>
<td>15</td>
<td>Primary key, identifies an individual tape.</td>
<td>N/A</td>
<td>Primary key</td>
<td>DDS</td>
</tr>
<tr>
<td>TAPE_POSITION</td>
<td>NUMBER</td>
<td>2</td>
<td>Primary key, identifies file position on tape.</td>
<td>N/A</td>
<td>Primary key</td>
<td>DDS</td>
</tr>
<tr>
<td>RAW_DATA_FILE_NAME</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Foreign key identifies raw data file.</td>
<td>N/A</td>
<td>Primary key, foreign key to DCS_RAWFILE_ACCT</td>
<td>DDS</td>
</tr>
<tr>
<td>BACKUP_DATE</td>
<td>DATE</td>
<td>n/a</td>
<td>Date file was written to tape.</td>
<td>not null</td>
<td>not null</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 13: BACKUP_ACCT**

### A.14 TEMP_CTS_FILES

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW_DATA_FILE_NAME</td>
<td>VARCHAR2</td>
<td>512</td>
<td>Foreign key identifies raw data file.</td>
<td>N/A</td>
<td>Primary key</td>
<td>DDS</td>
</tr>
<tr>
<td>TRANSFERRED</td>
<td>VARCHAR2</td>
<td>3</td>
<td>Flag indicating file has already been transferred (derived from write permission of file mode on CTS).</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>FILE_SIZE</td>
<td>NUMBER</td>
<td></td>
<td>Data file size in bytes</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 14: TEMP_CTS_FILES**
### A.15 DCS_CAPT_CHAN_MAP

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Default</th>
<th>Restrictions</th>
<th>Populated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPT_SYS_ID</td>
<td>VARCHAR2</td>
<td>20</td>
<td>Foreign key identifies the capture system.</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>CAPT_DEV_SLOT</td>
<td>VARCHAR2</td>
<td>50</td>
<td>Unix file system path for the Myriad Capture Device slot number.</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
<tr>
<td>CAPT_SRC_CHAN</td>
<td>VARCHAR2</td>
<td>2</td>
<td>Capture Channel Id (or capture source)</td>
<td>N/A</td>
<td>Not null</td>
<td>DDS</td>
</tr>
</tbody>
</table>

**TABLE 15: DCS_CAPT_CHAN_MAP**
### Appendix B  Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCI</td>
<td>Computer Software Configuration Item</td>
</tr>
<tr>
<td>DBDD</td>
<td>Database Design Description</td>
</tr>
<tr>
<td>SDS</td>
<td>Software Design Description</td>
</tr>
<tr>
<td>SDP</td>
<td>Software Development Plan</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirements Specification</td>
</tr>
</tbody>
</table>
Referenced Documents

USGS, DCS-100, Data Capture System (DCS) System Requirements Document (SRD), June 2000.
USGS, GS-100, Operations Concept, April 2000

NASA/GSFC, 511-4ICD/0296, Landsat 7 Mission Operations Center (MOC) to Landsat 7 Ground Station (LGS) Interface Control Document, Revision 2, October 1997

NASA/GSFC, LPS-104, Memorandum of Understanding between the Landsat 7 Processing System (LPS) and the Landsat 7 Mission Operations Center (MOC), June 1997

