

305-EED-001, Rev. 03

# **EOSDIS Evolution and Development (EED) Contract**

## **Release 8.3 Segment/Design Specifications for the EED Contract**

Revision 03

September 2014

Raytheon Company  
Riverdale, Maryland

This page intentionally left blank.

**Release 8.3  
Segment/Design Specifications  
for the EED Contract**

**Revision 03**

**September 2014**

Prepared Under Contract NNG10HP02C  
CDRL Item #023

**RESPONSIBLE ENGINEER**

 9/18/14  
Evelyn Nakamura, Chief Engineer Date  
EOSDIS Evolution and Development (EED) Contract

**SUBMITTED BY**

 9/18/14  
Kristene Rodrigues, Task Lead Date  
EOSDIS Evolution and Development (EED) Contract

**Raytheon Company**  
Riverdale, Maryland

This page intentionally left blank.

# Preface

---

This document is a formal contract deliverable. It requires Government review and approval within 45 business days. Changes to this document will be made by document change notice (DCN) or by complete revision.

Any questions or proposed changes can be addressed to:

Data Management Office  
The EED Contract Office  
Raytheon Company  
5700 Rivertech Court  
Riverdale, MD 20737

## Revision History

Document Number	Status/Issue	Publication Date	CCR Number
305-EED-001	Original	March 2011	11-0023
305-EED-001	Revision 01	April 2012	12-0091
305-EED-001	Revision 02	September 2013	13-0226
305-EED-001	Revision 03	September 2014	14-0272

This page intentionally left blank.

# Abstract

---

The Release 8.3 Segment/Design Specification is an overview description of the EED Project. The functionality of the ECS software is described at the Subsystem, Computer Software Configuration Item (CSCI), Computer Software Component (CSC), and Process levels. Architecture and context diagrams illustrate the process interconnections within the ECS CSCIs and the external connections to other CSCIs, subsystems, and specified segment interfaces. Interface event description tables describe the data, messages, notifications, or status information that occurs at each level of functionality within the ECS. A basic description of the Commercial Off The Shelf (COTS) software and hardware used in ECS is included.

The high-level design in this document is the level of information derived from requirement sources, and used by the development team to complete the ECS design implementation for a software system at an 8.3 state of maturity.

**Keywords:** Release 8.3, Overview, SDPS, CSMS, Design, Detailed Design, Subsystem, Architecture, Software, Hardware, Object Oriented, Security, Gateway, Reports, User Interface and GUI.

This page intentionally left blank.

# Contents

---

## 1. Introduction

1.1	Purpose and Scope .....	1-1
1.2	Document Organization .....	1-1

## 2. Related Documentation

2.1	Parent Documents .....	2-1
2.2	Applicable Documents .....	2-1
2.2.1	Other EED Related Documents and Documentation .....	2-1
2.2.2	Other ESDIS Related Documents and Documentation .....	2-2
2.3	ECS Tool Descriptions .....	2-2
2.3.1	ClearCase Baseline Manager Configuration Management Tool .....	2-2
2.3.2	ClearCase Delivery Tool .....	2-4
2.3.3	Configuration Change Request (CCR) Tool .....	2-6
2.3.4	Engineering Software (HOTSHELF) Tool .....	2-6
2.3.5	Passwords Maintenance Tool .....	2-7
2.3.6	Technical Documents Replication Tool .....	2-7

## 3. System Description

3.1	Mission and Release 8.3 Objectives .....	3-1
3.1.1	Release 8.3 Capabilities .....	3-1
3.2	Release 8.3 Architecture Overview .....	3-6
3.2.1	Release 8.3 Context Description .....	3-10
3.2.2	Release 8.3 Architecture .....	3-11

## 4. Subsystem Description

4.1	Data Server Subsystem Overview .....	4-11
4.1.1	Archive Inventory Management Software Description .....	4-17

4.1.2	DSS Error Handling and Processing .....	4-45
4.1.3	DSS Data Stores.....	4-46
4.2	DPL Ingest Subsystem Overview .....	4-48
4.2.1	DPL Ingest Computer Software Configuration Item Description.....	4-50
4.3	Client Subsystem Overview (DELETED) .....	4-61
4.4	Data Management Subsystem Overview .....	4-61
4.4.1	ECHO WSDL Order Component Software Description .....	4-64
4.4.2	Data Management Subsystem Hardware .....	4-70
4.5	Order Manager Subsystem Overview .....	4-72
4.5.1	Order Manager Subsystem Software Description.....	4-75
4.6	Communications Subsystem Overview .....	4-83
4.6.1	The Distributed Computing Configuration Item Software Description.....	4-85
4.6.2	The Distributed Computing Configuration Item Context .....	4-100
4.6.3	Distributed Computing Configuration Item Architecture .....	4-102
4.6.4	Distributed Computing Configuration Item Process Descriptions .....	4-102
4.6.5	Distributed Computing Configuration Item Process Interface Descriptions ...	4-102
4.6.6	Distributed Computing Configuration Item Data Stores .....	4-103
4.6.7	Communications Subsystem Hardware CI Description.....	4-103
4.7	Internetworking Subsystem (ISS) Overview.....	4-104
4.7.1	Internetworking Subsystem Description .....	4-105
4.7.2	Internetworking Hardware HWCI (INCI).....	4-107
4.8	EED General Process Failure Recovery Concepts.....	4-109
4.8.1	Client-Server Rebinding .....	4-109
4.8.2	Database Reconnecting.....	4-109
4.8.3	Request Identification .....	4-110
4.8.4	Request Responsibility.....	4-110
4.8.5	Queues.....	4-111
4.8.6	Request Responses.....	4-111
4.8.7	Duplicate Request Detection.....	4-116
4.8.8	Server Crash and Restart.....	4-116
4.8.9	Client Crash and Restart .....	4-118
4.9	Spatial Subscription Server (SSS) Subsystem Overview .....	4-120
4.9.1	Spatial Subscription Server Architecture.....	4-121

4.10	Data Pool Subsystem Overview.....	4-127
4.10.1	Data Pool Subsystem Context.....	4-129
4.10.2	Data Pool Hardware Context .....	4-132
4.10.3	Data Pool Insert CSCI Functional Overview .....	4-132
4.10.4	Data Stores .....	4-141
4.11	Bulk Metadata Generation Tool Subsystem Overview.....	4-142
4.11.1	BMGT Subsystem Context .....	4-142
4.11.2	BMGT/ECHO Interface .....	4-144
4.11.3	ECS Events and BMGT products .....	4-145
4.11.4	BMGT Architecture .....	4-148
4.11.5	Use of COTS in the BMGT Subsystem .....	4-151
4.11.6	BMGT Subsystem Software Description.....	4-151

## List of Figures

Figure 2.3-1.	ECS Baseline Information System Home Page.....	2-3
Figure 2.3-2.	ClearCase BLM Graphical User Interface.....	2-4
Figure 2.3-3.	ClearCase Delivery Tool Graphical User Interface.....	2-4
Figure 2.3-4.	Custom Software Tracking Page .....	2-5
Figure 2.3-5.	COTS Software Tracking Page .....	2-5
Figure 2.3-6.	CCR Tool Graphical User Interface.....	2-6
Figure 2.3-7.	Engineering Software (HOTSHELF) Page .....	2-7
Figure 3.2-1.	Example Hierarchical Software Diagram.....	3-9
Figure 3.2-2.	Release 8.3 Context Diagram.....	3-10
Figure 3.2-3.	Subsystem Architecture Diagram.....	3-12
Figure 4.1-1.	Data Server Subsystem Context Diagram .....	4-12
Figure 4.1-2.	AIM CSCI Context Diagram (DPLIngest).....	4-21
Figure 4.1-3.	AIM Interfaces with DAAC Operators (ESDT Maintenance GUI and QA Update utility).....	4-26
Figure 4.1-4.	AIM Interfaces with DAAC Operators (XML Replacement Utility).....	4-32
Figure 4.1-5.	AIM Interfaces with DAAC Operators (Granule Deletion Utilities) .....	4-34
Figure 4.1-6.	AIM Interfaces with DAAC Operators (Archive Check Utilities).....	4-37

Figure 4.1-7. AIM Interfaces with BMGT .....	4-41
Figure 4.1-8. AIM Context Diagram (OMS and DPL) .....	4-43
Figure 4.2-1. DPL Ingest Subsystem Context Diagram.....	4-48
Figure 4.2-2. DPL Ingest CSCI Context Diagram .....	4-51
Figure 4.4-1. Data Management Subsystem Context Diagram.....	4-62
Figure 4.4-2. ECHO WSDL Order Component CSCI Context Diagram .....	4-65
Figure 4.4-3. ECHO WSDL Order Component CSCI Architecture Diagram .....	4-67
Figure 4.5-1. Order Manager Subsystem Context Diagram.....	4-73
Figure 4.5-2. Order Manager Server CSCI Context Diagram.....	4-76
Figure 4.5-3. Order Manager Server CSCI Architecture Diagram .....	4-78
Figure 4.6-1. Virtual Terminal Context Diagram .....	4-87
Figure 4.6-2. Virtual Terminal Architecture Diagram .....	4-88
Figure 4.6-3. Cryptographic Management Interface Context Diagram .....	4-90
Figure 4.6-4. Cryptographic Management Interface Architecture Diagram .....	4-91
Figure 4.6-5. Domains Hierarchy Diagram.....	4-93
Figure 4.6-6. DNS Domains of the EED Project Diagram .....	4-94
Figure 4.6-7. ECS Topology Domains Diagram.....	4-94
Figure 4.6-8. Domain Name Server Context Diagram .....	4-95
Figure 4.6-9. Distributed Computing Configuration Item (DCCI) CSCI Context Diagram .....	4-101
Figure 4.7-1. DAAC Networks: Generic Architecture Diagram .....	4-105
Figure 4.9-1. Spatial Subscription Server Context Diagram .....	4-120
Figure 4.9-2. Spatial Subscription Server Architecture Diagram .....	4-122
Figure 4.10-1. Data Pool Subsystem Context Diagram.....	4-130
Figure 4.10-2. Data Pool Hardware Context.....	4-132
Figure 4.10-3. Data Pool Insert CSCI Architecture Diagram – Registration.....	4-133
Figure 4.10-4. Data Pool Insert CSCI Architecture Diagram – Publication .....	4-134
Figure 4.11-1. BMGT Subsystem High Level Context Diagram .....	4-143
Figure 4.11-2. BMGT Architecture Diagram .....	4-148

Figure 4.11-3. Dispatcher Using Producer-Consumer Design Pattern .....	4-153
Figure 4.11-4. Bucket on a Queue .....	4-156
Figure 4.11-5. Dispatcher Thread and the Sequence of Calls to Various Components.....	4-157
Figure 4.11-6. Component Dependencies.....	4-160
Figure 4.11-7. Dispatcher Database Sequence.....	4-161
Figure 4.11-8. Components of Exporter and Their Dependencies.....	4-166
Figure 4.11-9. Combined Sequences of Export Request and Export Activity Statuses .....	4-172
Figure 4.11-10. Manual Export Process Database Sequence.....	4-182

## List of Tables

Table 4-1. Memory Management Table .....	4-6
Table 4.1-1. Data Server Subsystem Interface Events .....	4-14
Table 4.1-2. AIM Software Components.....	4-18
Table 4.1-3. AIM Interfaces with DPLIngest.....	4-22
Table 4.1-4. AIM Interfaces with DAAC Operators (ESDT GUI, QA Update).....	4-27
Table 4.1-5. AIM Interfaces with DAAC Operators (XML Replacement Utility) .....	4-32
Table 4.1-6. AIM Interfaces with DAAC Operators (Granule Deletion) .....	4-35
Table 4.1-7. AIM Interfaces with DAAC Operators (Archive Check Utilities) .....	4-38
Table 4.1-8. AIM Interfaces with BMGT .....	4-41
Table 4.1-9. AIM Interfaces with OMS and DPL.....	4-43
Table 4.1-10. AIM CSCI Data Stores .....	4-46
Table 4.2-1. DPL Ingest Subsystem Interface Events .....	4-49
Table 4.2-2. DPL Ingest CSCI Interface Events .....	4-52
Table 4.2-3. DPL Ingest CSCI Processes.....	4-54
Table 4.2-4. DPL Ingest CSCI Process Interface Events .....	4-55
Table 4.2-5. DPL Ingest CSCI Data Stores.....	4-60
Table 4.4-1. Data Management Subsystem Interface Events.....	4-63
Table 4.4-2. ECHO WSDL Order Component CSCI Interface Events .....	4-66
Table 4.4-3. EWOC CSCI Processes .....	4-68

Table 4.4-4. EWOC CSCI Process Interface Events .....	4-68
Table 4.4-5. ECHO WSDL Order Component CSCI Data Stores.....	4-70
Table 4.5-1. Order Manager Subsystem Interface Events.....	4-73
Table 4.5-2. Order Manager Server CSCI Interface Events.....	4-76
Table 4.5-3. OMSRV CSCI Process .....	4-78
Table 4.5-4. Order Manager Server CSCI Process Interface Events .....	4-79
Table 4.5-5. CSCI Data Stores.....	4-82
Table 4.6-1. Communications Subsystem (CSS) Interface Events.....	4-83
Table 4.6-2. Virtual Terminal Interface Events .....	4-88
Table 4.6-3. Virtual Terminal Processes.....	4-89
Table 4.6-4. Virtual Terminal Process Interface Events .....	4-89
Table 4.6-5. Cryptographic Management Interface Events .....	4-90
Table 4.6-6. Cryptographic Management Interface Processes.....	4-91
Table 4.6-7. Cryptographic Management Interface Process Interface Events .....	4-92
Table 4.6-8. Cryptographic Management Interface Data Stores .....	4-92
Table 4.6-9. Domain Name Server Process .....	4-95
Table 4.6-10. Domain Name Server Process Interface Events .....	4-96
Table 4.6-11. Domain Name Server Data Stores.....	4-96
Table 4.6-12. Infrastructure Libraries .....	4-96
Table 4.6-13. Infrastructure Libraries Group Interfaces .....	4-98
Table 4.6-14. Distributed Computing Configuration Item (DCCI) CSCI Interface Events....	4-101
Table 4.6-15. CSMS CSCI to CSS CSC Mappings.....	4-102
Table 4.7-1. Internetworking Subsystem Baseline Documentation List.....	4-107
Table 4.7-2. Networking Hardware for EED Networks .....	4-107
Table 4.8-1. Request Responses .....	4-113
Table 4.8-2. Fault Handling Policies .....	4-114
Table 4.8-3. Server Response versus Restart Temperature.....	4-117
Table 4.8-4. Server Response for Request Re-submission .....	4-118
Table 4.8-5. Server Responses to Client Failures .....	4-119

Table 4.9-1. Subscription Server Interface Events.....	4-121
Table 4.9-2. Spatial Subscription Server Processes.....	4-123
Table 4.9-3. Spatial Subscription Server Process Interface Events .....	4-124
Table 4.10-1. Data Pool Subsystem Interface Events .....	4-130
Table 4.10-2. Use Cases for Data Pool Insert .....	4-134
Table 4.10-3. Data Pool Insert CSCI Process Description.....	4-135
Table 4.10-4. Data Pool ECS Insert CSCI Process Interface Events.....	4-137
Table 4.10-5. Data Pool Data Stores.....	4-141
Table 4.11-1. BMGT Subsystem High Level Interface Events .....	4-144
Table 4.11-2. Request Content Types .....	4-145
Table 4.11-3. Response Content Types .....	4-145
Table 4.11-4. ECS Event to BMGT Product Mapping.....	4-146
Table 4.11-5. BMGT Processes .....	4-149
Table 4.11-6. Metadata Native to Target Schema Mappings.....	4-163
Table 4.11-7. Generator Behavior.....	4-164
Table 4.11-8. ECHO Error Messages for Both Collections and Granules.....	4-177
Table 4.11-9. ECHO Error Messages for Both Collections and Granules.....	4-178
Table 4.11-10. ECHO Error Messages for Granules Only .....	4-179
Table 4.11-11. Data Store .....	4-187

## **Abbreviations and Acronyms**

This page intentionally left blank.

# 1. Introduction

---

## 1.1 Purpose and Scope

The purpose of the Segment/Design Specification for the Earth Observing System (EOS) Data and Information System (EOSDIS) Core System (ECS) is to provide an overview of the hardware and software subsystems of the project. This document describes the high-level design of each ECS software subsystem implemented to satisfy the allocated and derived functional and performance requirements. This document also provides basic descriptions of the Commercial Off The Shelf (COTS) hardware and software used in the ECS. This document contains:

- Functional overviews of each Computer Software Configuration Item (CSCI)
- Context diagrams of each CSCI
- Interface event descriptions based on the context diagrams
- Process architecture diagrams
- Interface event description tables based on the process architecture diagrams
- CSCI data stores (databases as they relate to the process architecture diagrams)
- CSCI functions allocated to processes. For data servers, this includes descriptions of the functionality offered to clients via the server interfaces. For Graphical User Interface (GUI) applications, it describes the functionality provided to the GUI users
- Specific limitations of the capabilities provided
- Abbreviations and Acronyms

## 1.2 Document Organization

The remainder of this document is organized as follows:

- Section 2: Related Documentation
- Section 3: System Description
- Section 4: Subsystem Description
- Section 5: Limitations of Current Implementation
- Abbreviations and Acronyms

This page intentionally left blank.

## 2. Related Documentation

---

### 2.1 Parent Documents

The parent documents are the documents from which the scope and content of this Design Specification are derived. These documents are listed below.

423-46-01	EED F&PRS
April 6, 2010	EED Task 02 Statement of Work For Providing ECS/ECHO Sustaining Engineering and Continuous Evolution

### 2.2 Applicable Documents

Refer to the 900 Series documentation found on the EED Baseline Information System (EBIS) websites:

<a href="http://pete.edn.ecs.nasa.gov/baseline/">http://pete.edn.ecs.nasa.gov/baseline/</a>	(Riverdale EDF Primary site)
<a href="http://ebis.gsfc.nasa.gov:10160/baseline/">http://ebis.gsfc.nasa.gov:10160/baseline/</a>	(ESDIS replicated site)
<a href="http://e5iil01u.cr.usgs.gov:10160/baseline/">http://e5iil01u.cr.usgs.gov:10160/baseline/</a>	(LP DAAC replicated site)
<a href="http://15iil01.larc.nasa.gov:10160/baseline/">http://15iil01.larc.nasa.gov:10160/baseline/</a>	(ASDC replicated site)
<a href="http://n5iil01u.ecs.nsidc.org:10160/baseline/">http://n5iil01u.ecs.nsidc.org:10160/baseline/</a>	(NSIDC replicated site)

#### 2.2.1 Other EED Related Documents and Documentation

Refer to the Section 1.2 “Applicable Documents” of the EED Task 02 Statement of Work For Providing ECS/ECHO Sustaining Engineering and Continuous Evolution.

311-EED-001, Rev. 03	Release 8.3 Ingest Subsystem (INS) Database Design and Schema Specifications for the EED Contract
311-EED-002, Rev. 03	Release 8.3 Order Manager Subsystem (OMS) Database Design and Schema Specifications for the EED Contract
311-EED-003, Rev. 03	Release 8.3 Spatial Subscription Server Subsystem (SSS) Database Design and Schema Specifications for the EED Contract
311-EED-005, Rev. 03	Release 8.3 Archive Inventory Management (AIM) Database Design and Schema Specifications for the EED Contract
611-EED-001, Rev. 03	Release 8.3 Mission Operations Procedures for the EED Contract
625-EED-001, Rev. 03	Release 8.3 Training Material Volume 1: Course Outlines
625-EED-002, Rev. 03	Release 8.3 Training Material Volume 2: Problem Management

- 625-EED-003, Rev. 03 Release 8.3 Training Material Volume 3: Ingest  
625-EED-004, Rev. 03 Release 8.3 Training Material Volume 4: Data Distribution  
625-EED-005, Rev. 03 Release 8.3 Training Material Volume 5: Archive Processing

## 2.2.2 Other ESDIS Related Documents and Documentation

Refer to the section 1.2 “Applicable Documents” of the EED Task 02 Statement of Work For Providing ECS/ECHO Sustaining Engineering and Continuous Evolution.

## 2.3 ECS Tool Descriptions

### 2.3.1 ClearCase Baseline Manager (BLM) Configuration Management Tool

The ClearCase Baseline Manager (BLM) tool produces the ECS baseline. Baseline visibility is provided by a set of ECS Baseline Information System (EBIS) file systems and a set of secure Apache web pages shown below:

<http://pete.edn.ecs.nasa.gov/baseline/>

<http://ebis.gsfc.nasa.gov:10160/baseline/>

<http://e5iil01u.cr.usgs.gov:10160/baseline/>

<http://15iil01.larc.nasa.gov:10160/baseline/>

<http://n5iil01u.ecs.nsidc.org:10160/baseline/>

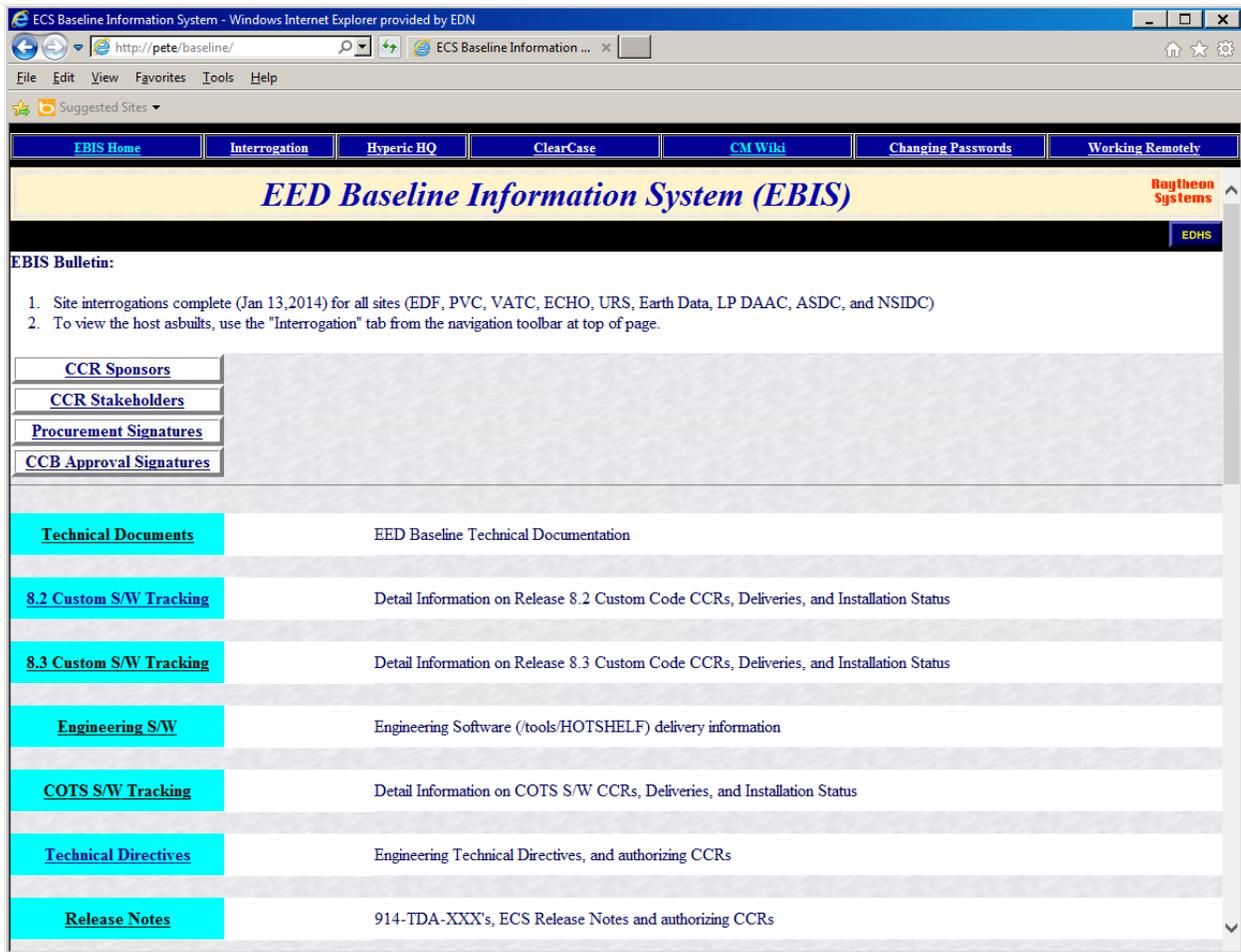
The EBIS file system is also viewable at the Riverdale facility on the Orion network at:

V:\Core\EED\EBIS\baseline\

Access control to the file systems is managed dependent on the site. Anyone with EDF credentials may view the “pete” URL above. This is the primary repository. The “ebis” URL is managed by firewall rules using IP addresses. Three ESDIS personnel may currently view this site. The “e5iil01u”, “15iil01”, and “n5iil01u” sites’ access are managed by the LP DAAC, ASDC, and NSIDC system administrators. As these sites are behind firewalls only those with local DAAC credentials may view these URLs. For convenience the EBIS file system is also replicated onto a local shared drive (V:) on the Raytheon Orion network for direct Riverdale access. Access is controlled by Raytheon personnel within the EED folder security setting.

Figure 2.3-1, *ECS Baseline Information System Home Page*, is shown on the next page.

This home page is present on all six EBIS URLs mentioned above. Selection of the embedded hyperlinks will yield subsequent pages which provide more detail information.



**Figure 2.3-1. ECS Baseline Information System Home Page**

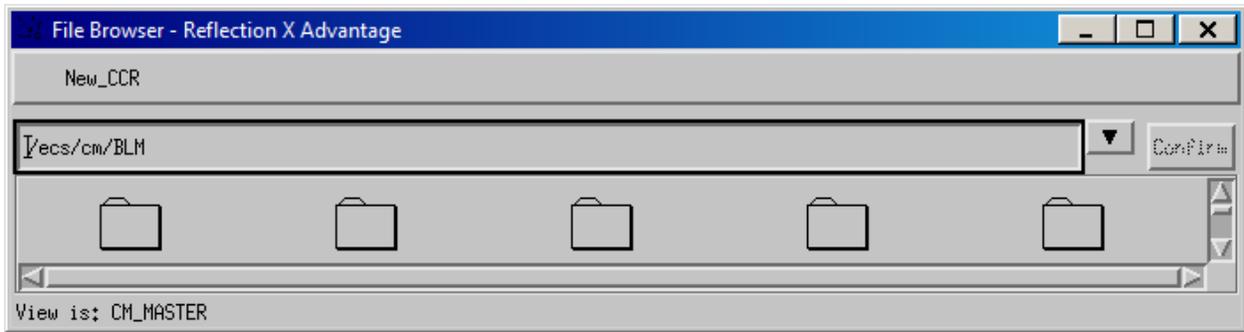
Changes to the EBIS file system is managed by an approved Change Control Request (CCR).

- 1) Release Notes (914-TDA-xxx) “Machines Affected” Section 6
- 2) An approved CCRs’ Installation Instruction

Once the BLM tool has promoted the updates to the Riverdale server they are replicated to the other 5 EBIS sites.

The BLM tool manages 34 EBIS technical documents consisting of over 300,000 lines of information for the EDF, PVC, and VATC at Riverdale, ECHO, Earthdata, and URS at GSFC, and LP DAAC, ASDC, and NSIDC at the DAAC sites.

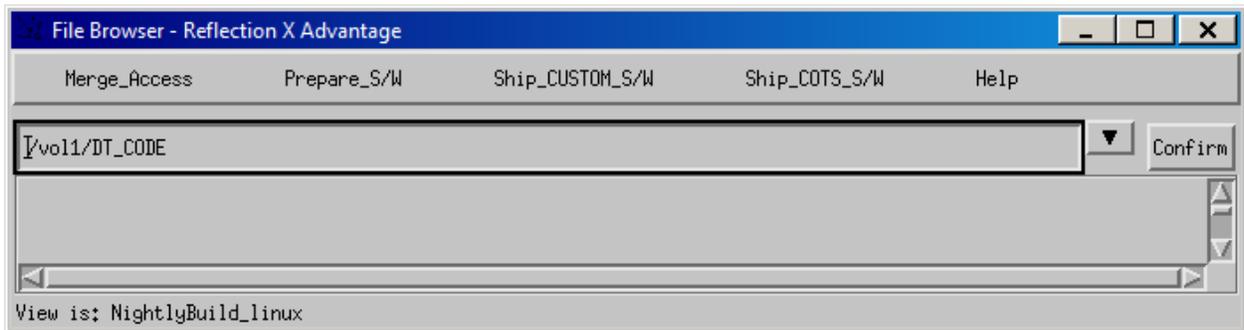
Figure 2.3-2, *ClearCase BLM Graphical User Interface*, is shown on the next page. The BLM tool facilitates ECS baseline updates and replications of the primary Riverdale EBIS information to the other 5 EBIS sites.



**Figure 2.3-2. ClearCase BLM Graphical User Interface**

### 2.3.2 ClearCase Delivery Tool

The ClearCase Delivery tool is used to prepare custom software tar files, assemble COTS software files, and deliver COTS and custom software to the PVC, LP DAAC, ASDC, and NSIDC. Delivery visibility is provided by the set of ECS Baseline Information System web pages.



**Figure 2.3-3. ClearCase Delivery Tool Graphical User Interface**

Figure 2.3-3 shown above is used to perform all functions of software preparations, custom software deliveries, as well as COTS software deliveries. The tool sends email to pertinent staff for each action. Updates to the custom and COTS software pages on each of the six EBIS file systems are also performed.

Figure 2.3-4, *Custom Software Tracking Page*, is shown on the next page. All relevant details are provided for each custom software delivery. Hyperlinks can be traversed to view detail information including the CCR, Installation Instructions, NCRs, and delivered files.

**Release 8.3 Custom Software Deliveries and Installation**

(Fri Sep 12 14:15:46 EDT 2014)

From the EDN select URL: <http://pete.edn.ecs.nasa.gov/baseline/>, then select the Blue Button, "8.3 Custom S/W Tracking".

Preparation and Delivery									Installation Verification			
Riverdale									Remote Sites			
Patch	Custom Code Name	Preparation Date	Approval Date	Shipment Date	CCR	Install Instruct.	NCRs	Files	PVC	LPDAAC	ASDC	NSIDC
2 <sup>nd</sup>	<a href="#">PATCH_8.3_ESDT.01_ICEBRIDGE</a>	Sep 10	Sep 12	Sep 12	<a href="#">14-0281</a>	<a href="#">A</a>	<a href="#">8051932</a> <a href="#">8051883</a> <a href="#">8051839</a> <a href="#">8051822</a> <a href="#">8051800</a>	7	Sep 10			susp.
1 <sup>st</sup>	<a href="#">Release 8.3</a> (Reference 914-TDA-562 on Release Notes page)	Aug 18	Aug 27	Aug 27	<a href="#">14-0226</a>	none	324	7	Aug 18	susp.	susp.	susp.

**Figure 2.3-4. Custom Software Tracking Page**

Figure 2.3-5, *COTS Software Tracking Page*, is shown below. All relevant details are provided for each COTS software delivery. Hyperlinks can be traversed to view detail information including the CCR, Release Notes or Installation Instructions, and delivered files.

**COTS Software Delivery and Installation Tracking**

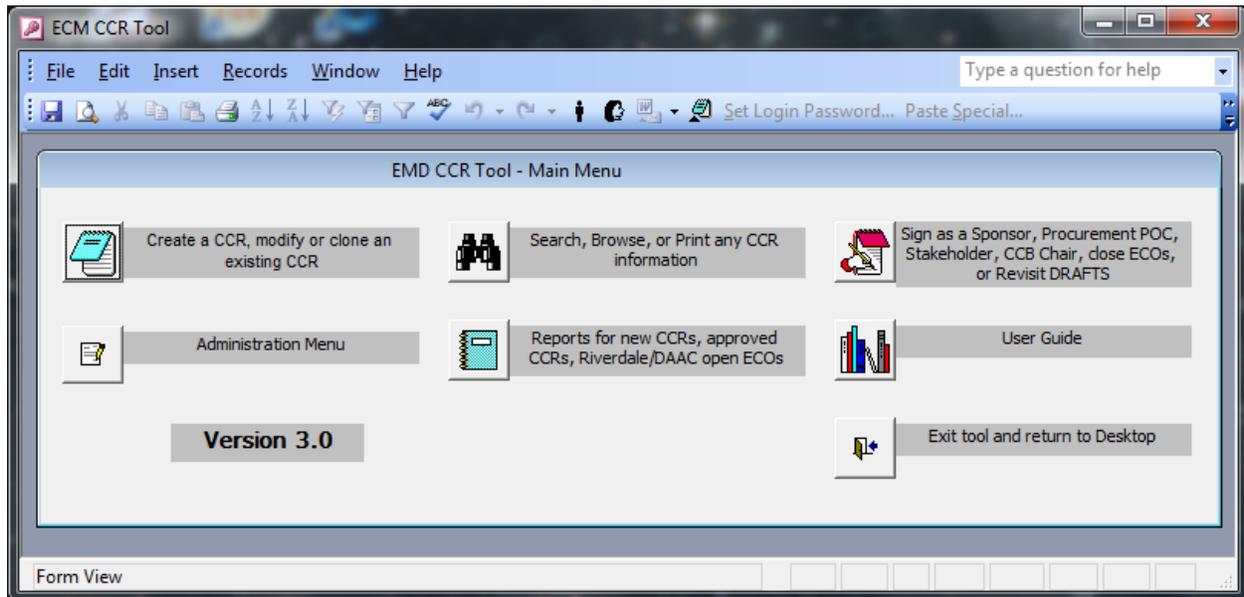
COTS Release	COTS S/W	CCR Approval Date	Date Sent	eCCR (html)	CCR (pdf)	Attachments (pdf/doc)	Files	Installation Verification Status						
								Riverdale			DAACs			
							VATC	PVC	SMC	LPDAAC	GES	ASDC	NSIDC	
678	<a href="#">StorNext 4.7.1</a>	Pending	Sep 4	<a href="#">14-0267</a>	none	<a href="#">914-TDA-563</a>	4				susp.		susp.	susp.
677	<a href="#">PostgreSQL 9.3.4</a>	Aug 1	Jul 31	<a href="#">14-0231</a>	none	<a href="#">914-TDA-561</a>	2				susp.		susp.	susp.
676	<a href="#">RHEL 5.10 and 6.4 with July 2014 patches and Firefox 24.6.0</a>	Aug 1	Jul 31	<a href="#">14-0227</a>	none	<a href="#">Attach.</a>	1				susp.		susp.	susp.
675	<a href="#">Apache 2.2.27 with SSL</a>	Jul 25	Jul 24	<a href="#">14-0217</a>	none	<a href="#">914-TDA-560</a>	2				susp.		susp.	susp.
674	<a href="#">OSSEC 2.8 and ClamAV 0.98.4</a>	Jul 18	Jul 18	<a href="#">14-0222</a>	none	<a href="#">914-TDA-559</a>	1				susp.		susp.	susp.
673	<a href="#">ClearCase 8.0.1.3-IFix01</a>	Jul 2	Jul 2	<a href="#">14-0195</a>	none	<a href="#">914-TDA-557</a>	2				susp.			

**Figure 2.3-5. COTS Software Tracking Page**

### 2.3.3 Configuration Change Request (CCR) Tool

The CCR Tool is used to facilitate changes to the ECS baseline as well as coordinate procurements.

Figure 2.3-6, *CCR Tool Graphical User Interface*, is shown below.



**Figure 2.3-6. CCR Tool Graphical User Interface**

### 2.3.4 Engineering Software (HOTSHELF) Tool

The Engineering Software (HOTSHELF) Tool is used to provide a 24x7 capability for certain ECS staff to send any number of files to remote sites. Each file can be of any size and type.

This mechanism ensures that immediate repairs can be made at LP DAAC, ASDC, and NSIDC. Local system administrators make the final determination to use the delivered software. The process requires an NCR number and README file as part of the delivery.

Email notifications are sent to pertinent staff regarding the deliveries and contents.

Figure 2.3-7, *Engineering Software (HOTSHELF) Page*, is shown below. By using the EBIS file systems and URLs it provides real-time visibility for each delivery.

2634 Engineering Software deliveries have been made to date. Each shows the Name, Sender, Transmit Time, README file, NCRs, delivered files, and the targets.

General Information								Targets						
Sequence Number	Custom Code	Sender	Transmit Date	Incorporation Date	README	NCRs /TTs	Files Sent	Riverdale			Remote Sites			
								PVC	VATC	SMC	LPDAAC	GES	ASDC	NSIDC
ES2634	<a href="#">IceBridge_Metgen_IODMS1B_Update_noBrowse</a>	mstanley	10:27 Sep 17	suspense	<a href="#">README</a>	<a href="#">1</a>	<a href="#">2</a>	-	-	-	-	-	-	lien
ES2633	<a href="#">8051786</a>	dclark	07:55 Sep 17	suspense	<a href="#">README</a>	<a href="#">1</a>	<a href="#">2</a>	-	-	-	-	lien	-	-
ES2632	<a href="#">IceBridge_IODMS1B.001_8</a>	jpan	15:37 Sep 11	suspense	<a href="#">README</a>	<a href="#">1</a>	<a href="#">4</a>	-	-	-	-	-	-	lien
ES2631	<a href="#">8051544_1</a>	rhartran	13:37 Sep 9	suspense	<a href="#">README</a>	<a href="#">1</a>	<a href="#">3</a>	lien	-	-	-	-	-	-
ES2630	<a href="#">IceBridge_IRTBE3_IRTBR3_IRTTE3.001</a>	jpan	14:09 Sep 5	suspense	<a href="#">README</a>	<a href="#">3</a>	<a href="#">6</a>	-	-	-	-	-	-	lien
ES2629	<a href="#">8051887</a>	dclark	12:41 Sep 2	suspense	<a href="#">README</a>	<a href="#">1</a>	<a href="#">1</a>	-	-	-	-	-	-	lien
ES2628	<a href="#">8051962_2</a>	jpan	18:05 Aug 28	suspense	<a href="#">README</a>	<a href="#">1</a>	<a href="#">1</a>	lien	-	-	-	-	-	-
ES2627	<a href="#">IceBridge_SMSM1B.001_Update_1_0014</a>	mstanley	10:50 Aug 25	suspense	<a href="#">README</a>	<a href="#">0</a>	<a href="#">0</a>	-	-	-	-	-	-	-

**Figure 2.3-7. Engineering Software (HOTSHELF) Page**

### 2.3.5 Passwords Maintenance Tool

The Password Maintenance Tool is used to maintain the 14 accounts and passwords that are needed for the CM tool scripts.

These are updated once per month using scripts that are encrypted and decrypted using OpenSSL.

### 2.3.6 Technical Documents Replication Tool

Technical documentation that is not managed by the ClearCase BLM tool must be replicated to the 5 remote EBIS file systems. A set of scripts exist to perform these functions.

These scripts also provide replication functions for non-procurement type CCRs and PreShip Reviews.

This page intentionally left blank.

## 3. System Description

---

### 3.1 Mission and Release 8.3 Objectives

The Mission of the National Aeronautics and Space Administration's (NASA) Earth Science Enterprise (ESE) is to develop a scientific understanding of the total Earth System and its response to natural or human-induced changes to the global environment to enable improved prediction capability for climate, weather and natural hazards. The vantage point of space provides information about Earth's land, atmosphere, ice, oceans and biota that is obtained in no other way. Programs of the enterprise study the interactions among these components to advance the new discipline of Earth System Science, with a near-term emphasis on global climate change. The research results contribute to the development of sound environmental policy and economic investment decisions.

The Earth Observing System Data and Information System (EOSDIS) Core System (ECS) has been designated as the ground system to collect, archive, produce higher-level data products and distribute data for the Earth System Science mission.

#### 3.1.1 Release 8.3 Capabilities

The ECS capabilities have been developed in increments called formal releases. Release 8.3, which is managed by Configuration Management, is a formal release. It is a collection of new and updated capabilities provided to the users of the system and is described here to show the progress of system enhancements. The ECS collects and stores, processes, archives and distributes scientific data from six different platforms (satellites). In the following sub-sections, the platforms and instruments from which scientific data is collected are identified, the type of data ingested and archived is presented, search and order capabilities for scientific data, how data is distributed and processed, system architecture and operation, system security and Distributed Active Archive Center (DAAC) and external system support are described. Other capabilities provided by Release 8.3 include distributing raw or processed data as requested, supporting communication networks, and systems monitoring via interfaces with the ECS operations staff. A summary of the highlights of Release 8.3 unique capabilities and modifications are as follows:

- GLAS HDF 5 subsetter
- Increased character limit for ECS shortnames
- Support for heterogenous multi-file granules for ASTL1T granules and full-resolution browse (FRB)

##### 3.1.1.1 ECS Support of Instruments by Platform

- The Meteor 3 platform supported the Stratospheric Aerosols and Gas Experiment III (SAGE III) instrument

- The ACRIMSAT platform supports the ACRIM III experiment
- The Terra (AM-1) platform supports the Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER), Multi-Angle Imaging SpectroRadiometer (MISR), Moderate Resolution Imaging SpectroRadiometer (MODIS) and Measurements of Pollution in the Troposphere (MOPITT) instruments
- The Aqua (PM-1) platform supports the Moderate Resolution Imaging SpectroRadiometer (MODIS) and Advanced Microwave Scanning Radiometer (AMSR-E) instruments
- The Ice, Cloud and Land Elevation satellite (ICESat) platform supported the Geoscience Laser Altimeter System (GLAS) instrument
- The Aura platform supports the Tropospheric Emission Spectrometer (TES) instrument
- Various aircraft support the set of Operation IceBridge instruments including the Airborne Topographic Mapper (ATM), the Digital Mapping System (DMS), and the Land, Vegetation, and Ice Sensor (LVIS), and the Pathfinder Advanced Radar Ice Sounder (PARIS).
- The SMAP (Soil Moisture Active Passive) platform will support the SMAP instrument.

### **3.1.1.2 Ingest and Archive Capabilities**

The ECS provides the following ingest and archive capabilities at each DAAC. Details on data types supported at each DAAC are described in 3.1.1.8.

- Ingest of L0, science and engineering data from the EOS Data and Operations System (EDOS)
- Ingest of higher level products (L1, L2, L3) from SIPS, SCFs and various sources
- Ingest of Product Generation Executable (PGE) software from Science Computing Facilities (SCFs) either electronically or via media tape
- Ingest of FDS (formerly FDD) orbit data

### **3.1.1.3 Search and Order Capabilities**

The ECS provides the following capabilities for search and ordering of data from the archive:

- Handling of orders from ECHO via EWOC
- CLS provides an user interface to query Order Status
- Handling of variations on search areas and product-specific spatial representations
- The SSS provides an operator the interface to place standing orders (subscriptions) based on an ECS event and manage subscription status
- The Data Pool provides an operator the interface to manage insert processes, queues, collection groups and collection themes for ECS and non-ECS collections

#### **3.1.1.4 Data Distribution Capabilities**

The ECS provides the following Data Distribution capabilities for users:

- Support distributing science data products via FTP Push or Pull, and SCP Push.

#### **3.1.1.5 Data Management Capabilities**

The ECS provides the following capabilities for user/operator data management options:

- Support the archive of products previously produced and archived
- Provide capability for operator deletion of granules
- Provide capability to associate the ASTER browse granule for the L1A product with ASTER L1B products

#### **3.1.1.6 System Operation and Architecture**

The ECS provides the following capabilities to support the system operations and processing architecture used to provide data and services for users:

- Provide capability for operator deletion of granules, their associated metadata and browse files
- Provide the associated communications network interfaces with the SCFs
- Support managing the startup and shutdown of system network components, database and archive administration, system data and file back-up and restores, system performance tuning and resource usage monitoring, and other routine operator duties
- Operations support to update certain ESDT attributes without requiring the deletion of the data collection
- Provide the capability for editing of ECS core attribute values
- Support the consolidation of trouble tickets using TestTrack Pro
- Provide fault recovery for mode management
- Provide the capability for startup and shutdown of an entire mode
- Provide the capability for the deletion of science data from the archive
- Provide the capability for the installation of ESDTs to insert and acquire archived data
- Provide for the storage of event information into the AIM database
- Provide the capability for the monitoring of the usage of memory
- Provide COTS packages to allow operations to generate customized reports from ECS databases

- Provide a single configuration registry database to replace the numerous ECS application configuration files
- Provide for the insertion of ECS granules into the Data Pool
- Export ECS metadata to ECHO via BMGT
- Provide delete, publish, restore, validation, consistency and integrity checking tools

### **3.1.1.7 Security**

The ECS provides the following capabilities for system security:

- Encryption of passwords in ECS databases
- SDP Toolkit support for thread safe concurrent processing by the science software
- Secure Transfer of data files from Data Providers upon request
- System data and file backups and restores

### **3.1.1.8 DAAC/External System Support**

ECS Release 8.3 will be distributed to three site locations including:

1. The Atmospheric Science Data Center (ASDC) DAAC at the Langley Research Center (LaRC),
2. The Land Processes DAAC (LP DAAC), and
3. The DAAC at the National Snow and Ice Center (NSIDC)

The ECS Release 8.3 communications network includes the National Aeronautics and Space Administration (NASA) and the NASA Integrated Services Network (NISN). These portions of the network are physically located at the DAAC sites. The communications network connects ECS to data providers at the EDOS, NOAA Affiliated Data Center (ADC), and the EOSDIS Version 0 system.

The data users for Release 8.3 are the science user community connected to the ECHO, Data Pool Webaccess, the three DAACs, the SCFs, and the MODAPS.

#### **1. LaRC Support:**

- ECS Release 8.3 provides a communications network and data/information management support for MISR instrument data including the receipt of MISR level 0 data and the LaRC archive and distribution of levels 1, 2 and 3 data and data products
- ECS Release 8.3 provides a communications network and data/information management support for MOPITT instrument data including the receipt of MOPITT level 0 data, the LaRC archive, and distribution of levels 1, 2 and 3 data

- ECS Release 8.3 provides a communications network and data/information management support for TES instrument data including the receipt of TES level 0 data and the LaRC archive, and distribution of levels 1, 2 and 3 data
- LaRC DAAC capabilities include:
  - Ingest of MISR, TES, and MOPITT Level 0 and related ancillary data
  - Archival, and distribution of the higher-level products for MISR
  - Receipt of higher-level MOPITT products from the MOPITT SCF, via the SIPS interface, for archival and distribution
  - Receipt of SAGE III products from the SCF, via the SIPS interface, for archival and distribution
  - Receipt of ACRIM products (Level 0 and Level 2 data) from the SCF, via the SIPS interface, for archival and distribution
  - Receipt of TES Levels 1-3 data including algorithm and associated software packages, metadata, production histories, ancillary data and Quality Assessment (QA) data for archival and distribution

## 2. LP DAAC Support:

- ECS Release 8.3 provides a communications network and data/information management support for ASTER instrument data including the receipt of ASTER level 1A data electronically at LP DAAC from Japan, and distribution of higher level ASTER products by LP DAAC
- LP DAAC capabilities include:
  - Ingest of ASTER Level 1A/1B, with ancillary data needed for production
  - Archival and distribution of ASTER products
  - Receipt of higher level MODIS land products from MODAPS, via the SIPS interface, for archival and distribution

## 3. NSIDC Support:

- AMSR-E instrument data including the receipt of level 0 data from EDOS at ECS, and the NSIDC archive and distribution of levels 1, 2 and 3 data. The Level 1A data is received from the NSIDC V0 DAAC while the level 2 and 3 data is received from the AMSR-E SCF via the SIPS interface
- AMSR-ADEOS II Level 1A data is received from the NSIDC DAAC and archived and distributed using ECS.
- ECS Release 8.3 supports the ingest of ICESat GLAS level 1, level 2, level 3 and ancillary input data for archive and distribution at the NSIDC DAAC using the standard SIPS interface.

- NSIDC DAAC capabilities include:
  - Receipt of higher-level MODIS snow and ice products from MODAPS, via the SIPS interface, for archival and distribution
  - Ingest of AMSR-E Level 0 data and related ancillary data
  - Receipt of the AMSR-E and AMSR-ADEOS II higher-level products via the SIPS interface, for archival and distribution
  - Archival and distribution of GLAS Level 0 data and related ancillary data
  - Receipt of the GLAS higher level products from the SCF, via the SIPS interface, for archival and distribution
  - Receipt of the Operation IceBridge data including ATM, DMS, LVIS, and PARIS higher-level products, among others, via the SIPS interface, for archival and distribution. The product metadata are created by the ECS MetGen program.
  - Receipt of SMAP Level 0 through Level 4 data from the SMAP SDS, via the SIPS interface, for archival and distribution

#### 4. SCF Support:

- The MOPITT higher-level products are generated at the SCF and provided to the ECS via the SIPS interface
- ECS Release 8.3 supports receiving SAGE III Level 0 data and higher level products from the SCF via the SIPS interface
- ECS Release 8.3 supports receiving ACRIM L0 data and higher level products from the SCF via the SIPS interface

#### 5. MODAPS Support

- ECS Release 8.3 provides a communications network and data/information management support for MODIS instrument data including: archive and distribution of higher level data from the MODIS Data Processing System (MODAPS)

### **3.2 Release 8.3 Architecture Overview**

The ECS Release 8.3 architecture comprises the logical items listed here. Commercial Off The Shelf (COTS) software and hardware are used, to the extent possible, to implement the ECS functionality of these logical items.

- System
- Segments
- Subsystems

- Computer software configuration items (CSCIs)
- Computer software components (CSCs)
- Processes

ECS Release 8.3 was built of the following two segments.

- CSMS – Communications and Systems Management Segment
- SDPS – Science Data Processing Segment

Each segment was in turn built of the following subsystems:

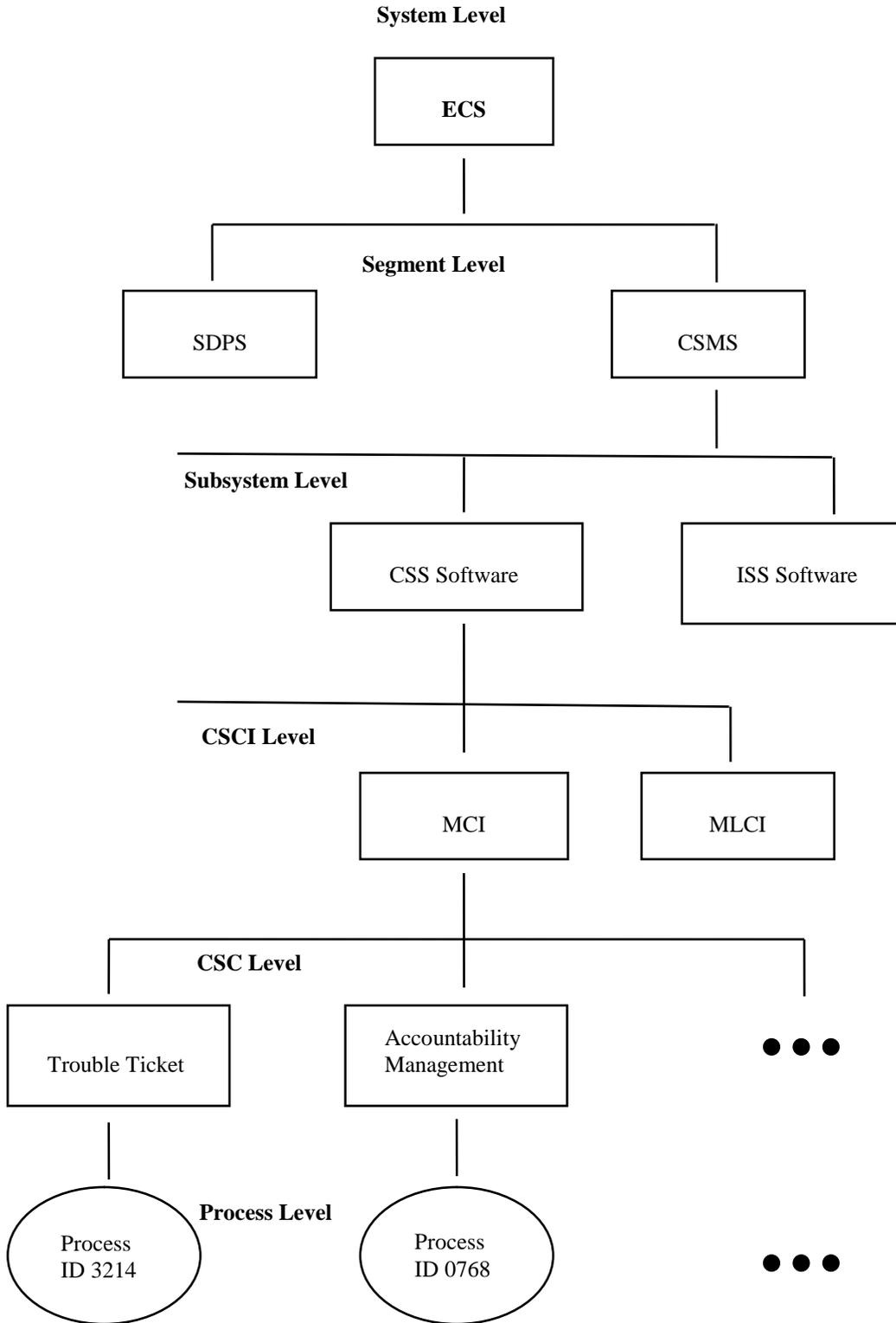
- CSMS: CSS – Communications Subsystem  
ISS – Internetworking Subsystem
- SDPS: BMGT – Bulk Metadata Generation Tool Subsystem  
CLS – Client Subsystem  
DMS – Data Management Subsystem  
DPL – Data Pool Subsystem  
DPL INGEST – Data Pool Ingest Subsystem  
DSS – Data Server Subsystem  
OMS – Order Management Subsystem  
SSS – Spatial Subscription Server Subsystem

### **Hierarchical Definitions**

- System:** A stand-alone composite of hardware, facilities, material, software, services, and personnel required for operation based upon a defined set of system level requirements and designed as a related set of capabilities and procedures.
- Segment:** A logical and functional subset of related capabilities, implemented with COTS hardware and COTS and custom developed software to satisfy a defined subset of the system level requirements.
- Subsystem:** A logical subset of Segment related capabilities, implemented with COTS hardware and COTS and custom developed software to satisfy a defined subset of segment level requirements.
- CSCI:** A logical subset of Subsystem related capabilities, implemented with COTS and custom developed software to satisfy a defined subset of the subsystem level software requirements.
- CSC:** A logical subset of CSCI related capabilities, implemented with COTS and custom developed software to satisfy a defined subset of the CSCI level software requirements.

Process: A logical and functional set of software, written in a specific order and in a defined manageable size to manipulate data as part of a product-generating algorithm. A process is a separately compiled executable (i.e., binary image). A process can use infrastructure library calls, system service calls, COTS service calls, and application programming interfaces to manipulate data to generate products.

Figure 3.2-1 is a hierarchical software diagram. The hierarchical software diagram depicts an example of the decomposition levels used in the ECS design and described in this document. The diagram is also a graphical representation of the terms just described.

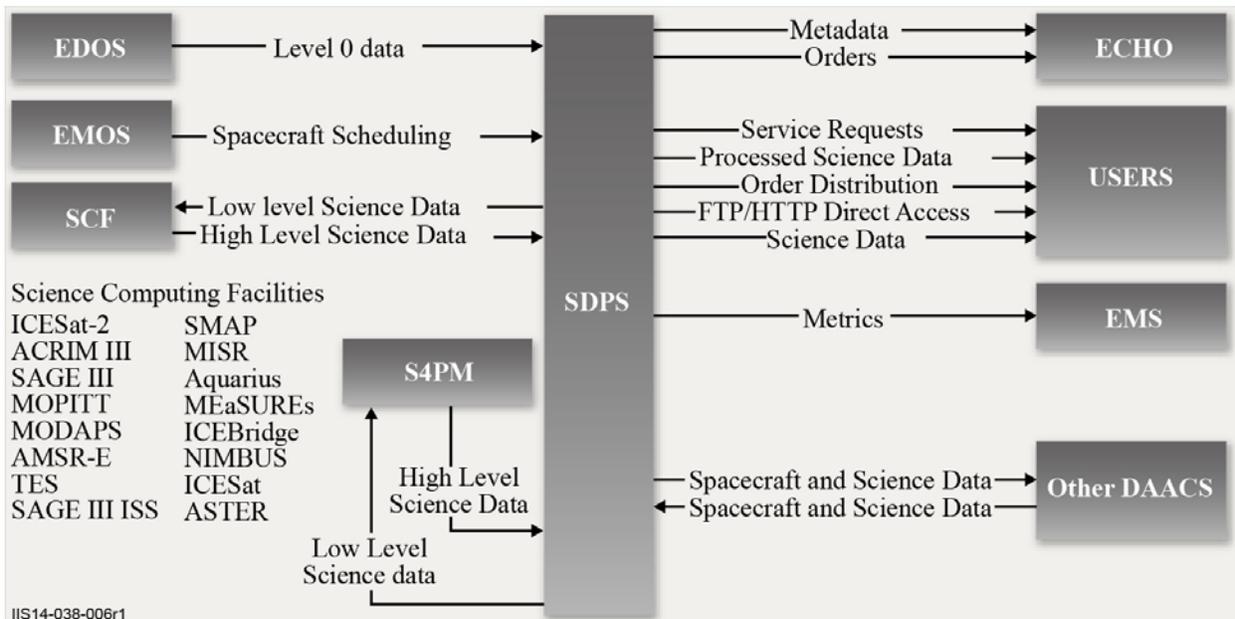


**Figure 3.2-1. Example Hierarchical Software Diagram**

### 3.2.1 Release 8.3 Context Description

ECS Release 8.3 provides the capability to collect and process satellite science data as depicted in Figure 3.2-2.

The Science Data Processing and Communications and Systems Management are the two segments of Release 8.3 described in this document. The Science Data Processing Segment (SDPS) provides science data ingest, search and access functions, data archive, and system management capabilities. The SDPS receives Level 0 science data from EDOS. The SDPS exchanges data with affiliated data centers to obtain science and other data (i.e., engineering and ancillary) required for data production. Science algorithms, provided by the Science Computing Facilities (SCFs), are archived for distribution. The Communications and Systems Management Segment (CSMS) provides the communications infrastructure for the ECS and systems management for all of the ECS hardware and software components. The CSMS provides the interconnection between users and service providers within the ECS, transfer of information between subsystems, CSCIs, CSCs, and processes of the ECS.



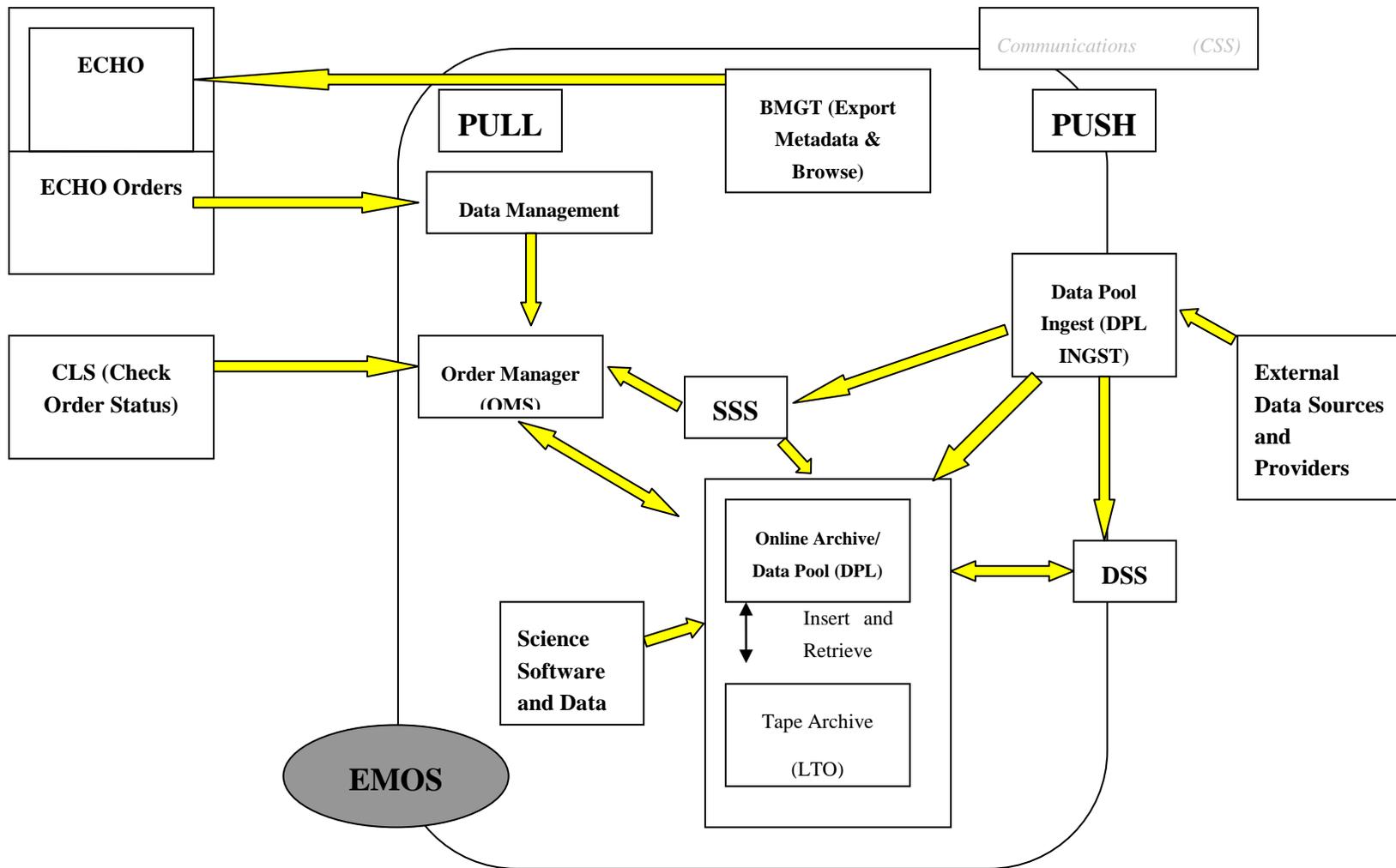
**Figure 3.2-2. Release 8.3 Context Diagram**

The remaining sections of this document provide an overview of Release 8.3 design and as such do not deal specifically with the configuration of components at each EOSDIS site. For more information on the site unique configurations, refer to the 920-series of General documents. Each of the segments consists of subsystems as specified in Section 3.2.

## **3.2.2 Release 8.3 Architecture**

### **3.2.2.1 Subsystem Architecture**

The ECS SDPS subsystems are depicted in Figure 3.2-3. A subsystem consists of the Commercial Off The Shelf (COTS) and/or ECS developed software and the COTS hardware needed for its execution. The SDPS subsystems can be grouped into a 'Push' or 'Pull' category of functionality with the exception of DSS. As shown in the subsystem architecture diagram, the information search and data retrieval makes up the 'Pull' side of the ECS architecture/design and consists of the CLS, DMS, OMS, SSS, DPL and also uses the DSS functionality described on the 'Push' side of the ECS architecture. Data capture (ingest of data), storage management, planning and data processing of satellite or previously archived data from other sites make up the 'Push' side of the ECS architecture/design and consists of the DSS, DPL, DPL INGEST, and OMS. This document describes the software and hardware components of each subsystem. However, since the hardware configurations differ between the sites, the hardware descriptions in this document are at a generic level. Specific hardware and network configurations for each site are documented in the 920 and 921 series technical documents.



**Figure 3.2-3. Subsystem Architecture Diagram**

The ECS SDPS architecture/design consists of:

- BMGT exporting inventory status information to ECHO.
- CLS providing a user interface to check order status.
- DMS providing support for data retrieval across all ECS sites.
- DPL supporting the search, order, and distribution of selected granules with associated metadata and browse granules (if available).
- DPL INGEST service will handle the SIPS ingest interface, S4P, cross-DAAC ingest, EDOS ingest, ASTER Ingest and Polling without Delivery Record specifically for EMOS.
- DSS with the functions needed to manage the inventory of archived data.
- OMS managing all orders received from the DMS EWOC and the Spatial Subscription Server.
- SSS supporting the creation, viewing and updating of subscriptions and the creation, viewing and deletion of bundling orders (specification of distribution packages and criteria for package completion).

CSMS – The following subsystems are the CSMS subsystems, which interact with and support the SDPS to complete the ECS architecture.

1. The CSS with:

- Control Center System (CCS) Middleware provides a common Name Server, which packages the common portions of the communication mechanisms into global objects to be used by all subsystems. The Name Server provides a set of standard CCS Proxy/Server classes, which encapsulates all of the common code for middleware communications (e.g., portals, couplers, etc.)
- Libraries with common software mechanisms for application error handling, aspects of recovering client/server communications; Universal References to distributed objects and interfaces to e-mail, file transfer and network file copy capabilities

2. The ISS with:

- Networking hardware devices (e.g., routers, switches, hubs, cabling, etc.) and their respective embedded software. For more information on site unique configurations, refer to the 920-series of General documents

This page intentionally left blank.

## 4. Subsystem Description

---

### Design Description Organization and Approach

This section presents a subsystem-by-subsystem overview description of the “as-built” ECS. The current high-level design information is provided for the Hardware Configuration Items (HWCI), Computer Software Configuration Items (CSCI), and Computer Software Components (CSC) for each subsystem and is being delivered to the DAACs in drop increments.

The CSMS subsystem descriptions include:

- Subsystem functional overviews with a subsystem context diagram and a table of interface event descriptions
- CSCI descriptions with a context diagram and a table with interface event descriptions
- Architecture Diagrams, Process Descriptions, and Process Interface Event Tables. The Architecture Diagrams show the processes of the CSCI/CSC and how these processes connect with other CSCIs and CSCs of the same subsystem and the interfaces with other subsystems and external entities such as Operations, External Data Providers and Users.
- Data Store descriptions for each CSCI in the CSMS subsystem. The Data Stores are identified with the software name and shown in the architecture diagrams either as single data stores or as a group of data stores with a generic name such as “Data Stores” or “database”
- Hardware descriptions of the subsystem hardware items and the fail-over strategy

The convention used for Context and Architecture diagrams includes using circular shapes to show the subject subsystems, CSCIs, CSCs, or processes (with name in bold), elliptical shapes to show associated CSCIs, CSCs, or processes within a given subsystem and squares or rectangles to show external subsystems, CSCIs, CSCs, and processes. Data stores are shown using the data store or database name with a pair of horizontal lines one above and one below the name, or as a cylinder with the name below. An interface event is data, a message (which includes a notification or status); a command, request or status code passed between subsystems, CSCIs, CSCs, or processes. The convention used to identify events is a straight line between two objects labeled with a phrase beginning with an action-oriented word to best describe the event. The arrow on the event line indicates an origination point and to where the event is directed. A direct response to an event is not always shown on the diagram because sometimes there is no response (e.g., for an insert or delete request) and other times the response comes from another part of the ECS. Interface events are identified in the interface event or process interface tables starting with the interface event at the top or middle of the diagram and going clock-wise around the diagram. The external interface subsystem is identified in the interface event description and is in bold to assist with the location of the interface events on the diagram. If there are two items in bold, there are two different interfaces (Subsystems, CSCIs, or CSCs) requesting the same interface event. These conventions are consistent with other ECS documentation. The convention for naming the ECS processes is *Ec* <subsystem abbreviation> meaningful name. The *Ec* identifies

the process as an ECS developed process versus a Commercial Off The Shelf (COTS) product. The *subsystem abbreviations* are listed subsystem-by-subsystem.

- Bm for BMGT
- Cl for CLS
- Cs for CSS
- Dl for DPL (also Data Pool Ingest and DataAccess)
- Dm for DMS
- Ds for DSS
- Ms for MSS
- Nb for SSS
- Om for OMS

The *meaningful name* identifies the process and its functionality within the subsystem, CSCI, or CSC. An example is EcDsAmXvu, which identifies an ECS-developed DSS process called the AIM XML Validation Utility. Some names within an architecture diagram do not follow this convention because the names are COTS product names. All COTS product names are kept for simplicity and to adhere to licensing and trademark agreements. The remaining names that do not follow the naming convention are imbedded throughout the system and would require time to replace and cause operational disruptions. These names will be cleaned up during the final maintenance stages of the contract if directed by the customer.

### **Object-oriented modeling and design**

*Object-oriented modeling and design* is a new way of thinking about problems using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity. Object-oriented models are useful for understanding problems, communicating with application experts, modeling enterprises, preparing documentation and designing programs and databases.<sup>1</sup>

Superficially the term "object-oriented" means that we organize software as a collection of discrete objects that incorporate both data structure and behavior. This is in contrast to conventional programming in which data structure and behavior are only loosely connected. There is some dispute about exactly what characteristics are required by an object-oriented approach, but generally include four aspects: identity, classification, polymorphism and inheritance.<sup>1</sup> *Identity* means that data is quantized into discrete, distinguishable entities called *objects*. A paragraph in my document, a window on my workstation and a white queen in a chess game are examples of objects. Objects can be concrete, such as a file, or conceptual, such as a *scheduling policy* in a multi-processing operating system. Each object has its own inherent

---

<sup>1</sup> Object-oriented Modeling and design, James Rumbaugh et al, copyright 1991 by Prentice-Hall, Inc. ISBN 0-13-629841-9

identity. In other words, two objects are distinct even if all their attribute values (such as name and size) are identical.<sup>1</sup>

In the real world an object simply exists, but within a programming language each object has a unique *handle* by which it can be uniquely referenced. The handle may be implemented in various ways, such as an address, array index or unique value of an attribute. Object references are uniform and independent of the contents of the objects, permitting mixed collections of objects to be created, such as a file system directory that contains both files and sub-directories.<sup>1</sup>

*Classification* means that objects with the same data structure (attributes) and behavior (operations) are grouped into a *class*. Paragraph, Window, and ChessPiece are examples of classes. A *class* is an abstraction that describes properties important to an application and ignores the rest. Any choice of classes is arbitrary and depends on the application.<sup>1</sup>

Each class describes a possibly infinite set of individual objects. Each object is said to be an instance of its class. Each instance of the class has its own value for each attribute but shares the attribute names and operations with other instances of the class. An object contains an implicit reference to its own class: it "knows what kind of a thing it is."<sup>1</sup>

*Polymorphism* means that the same operation may behave differently on different classes. The *move* operation, for example, may behave differently on the *Window* and *ChessPiece* classes. An *operation* is an action or transformation that an object performs or is subject to. *Right justify*, *display* and *move* are examples of operations. A specific implementation of an operation by a certain class is called a *method*. Because an object-oriented operator is polymorphic, it may have more than one method implementing it.<sup>11</sup>

In the real world, an operation is simply an abstraction of analogous behavior across different kinds of objects. Each object "knows how" to perform its own operations. In an object-oriented programming language, however, the language automatically selects the correct method to implement an operation based on the name of the operation and the class of the object being operated on. The user of an operation need not be aware of how many methods exist to implement a given polymorphic operation. New classes can be added without changing existing code, provided methods are provided for each applicable operation on the new classes.<sup>1</sup>

*Inheritance* is the sharing of attributes and operations among classes based on a hierarchical relationship. A class can be defined broadly and then refined into successively finer *subclasses*. Each sub-class incorporates, or *inherits* all the properties of its *super-class* and adds its own unique properties. The properties of the superclass need not be repeated. For example, *ScrollingWindow* and *FixedWindow* are subclasses of *Window*. Both subclasses inherit the properties of *Window*, such as a visible region on the screen.<sup>1</sup>

The ECS is a large, complex data storage and retrieval system used to store and retrieve large amounts of science and science-related data. The system was designed using an object oriented design approach. With so many objects and the sizes of some of them, it is necessary to have

---

<sup>1</sup> Object-oriented Modeling and design, James Rumbaugh et al, copyright 1991 by Prentice-Hall, Inc. ISBN 0-13-629841-9

some insight into the amount of memory being utilized within the ECS. The information about to be presented is a brief look at the memory management of the "key" (top ten utilized) objects within the ECS subsystems.

In this object oriented system design, objects are created and used via classes throughout the system to help perform the functions and meet the needs of the system. The objects for the ECS are very numerous, sometimes very large and cannot be provided in their entirety at this time. However, presented in the table below are the "key" objects for this system and how they are created, passed and deleted within the ECS.

### **Introduction to memory management approaches and memory usage table**

Good memory management in some applications is important and requires significant planning and development time. Many important ECS applications are large, long running, multi-threaded, heavy memory users and therefore are prime candidates for improved memory management.<sup>1</sup>

Improper memory management can result in memory leaks, fast memory usage growth or large application footprints and random crashes. ECS servers are periodically purified for memory leaks and there is a history of progress in this area. Similar work should be expected to continue as development and maintenance continues.

Long running server like applications that are free from memory leaks can nonetheless have significant memory and Central Processing Unit (CPU) usage performance degradation. A common culprit is heap fragmentation. The repeated allocation and deallocation of memory (such as with the new and delete operators of C++) can result in a large number of unusable free blocks of memory. They are free blocks but are interspersed with non-free blocks. They become unusable since they are not contiguous (fragmented) and as time goes by, it becomes harder and harder for the OS to service requests for more memory. Such situations even lead to crashes of other, non-offending applications running in the same box.

There are strategies, tools and software to avoid both memory leaks and fragmentation. This includes but is not limited to:

- Periodic application of purification software (already an ECS practice)
- Software design which uses dynamic memory as little as possible, such as automatic storage or COTS data structures
- Class-level memory management to allocate large chunks of memory instead of one class instance at a time ("Effective C++" by Scott Meyers and "Advanced C++" by James Coplien address this technique)
- Non-class level memory pools and
- COTS heap manager

---

<sup>1</sup> Object-oriented Modeling and design, James Rumbaugh et al, copyright 1991 by Prentice-Hall, Inc. ISBN 0-13-629841-9

Table 4-1 below is provided in case further memory management improvements are desired. Given operator or field input of seemingly inefficient memory or CPU usage, this table can be used to help target specific ECS subsystems, servers and frameworks or classes for improvement. It can be decided to apply some of the approaches at one level (e.g., on one guinea pig server or class) or perhaps experiment with changing the entire ECS C++ system with the use of a COTS heap manager. In any case, a great deal of planning and manpower is required.

**Table 4-1. Memory Management Table (1 of 5)**

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/process name) (M)	Passed To (Executable/process name)	Where Deleted? (process name) (M)	Number of Instances (Example – 1 per granule)	Comments/Remarks (Items of special interest. Example - Size per instantiation, never “deleted”, etc.)
<b>DPLINGEST</b>	EcDlInPollingService EcDlInProcessingService EcDlInNotificationService	DpCoAlert	Used to describe problematic conditions in DPLINGEST that require attention by the operator.	EcDlInPollingService EcDlInProcessingService EcDlInNotificationService	not passed	EcDlInPollingService EcDlInProcessingService EcDlInNotificationService	One per condition	Alerts are used to convey descriptions of problems to database and are maintained within the service for the lifetime of the problem. Deleted when alert is cleared by operator.
	EcDlInPollingService EcDlInProcessingService EcDlInNotificationService	DpCoMessage	Used to communicate a resource addition, modification, subtraction, suspension or resumption from the operator to the service.	EcDlInPollingService EcDlInProcessingService EcDlInNotificationService	not passed	EcDlInPollingService EcDlInProcessingService EcDlInNotificationService	One per operator action	Messages are used to communicate changes in resource state and are maintained within the service until the change has been made to the services representation of that resource.

**Table 4-1. Memory Management Table (2 of 5)**

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/process name) (M)	Passed To (Executable/process name)	Where Deleted? (process name) (M)	Number of Instances (Example – 1 per granule)	Comments/Remarks (Items of special interest. Example - Size per instantiation, never “deleted”, etc.)
	EcDInPollingService EcDInProcessingService EcDInNotificationService	DpCoResource and children.	Used to describe a resource used by service. For example, ftp hosts, providers.	EcDInPollingService EcDInProcessingService EcDInNotificationService	not passed	EcDInPollingService EcDInProcessingService EcDInNotificationService	One per resource.	Created on startup or when a resource is added to the system. Deleted when a resource is removed from the system or the service is shutdown.
	EcDInPollingService	DpInPoller	Used to obtain PDR files from a specific polling location	EcDInPollingService	not passed	EcDInPollingService	One per polling location	Created on startup. Deleted when a polling location is removed or the service is shutdown
	EcDInPollingService	DpInResourceCheckTimer	Used to periodically poll for changes in ingest resource properties.	EcDInPollingService	not passed	EcDInPollingService	One	Created on startup. Deleted on shutdown.
	EcDInPollingService	DpInPollingDatabase	Used to handle all interaction with ingest database	EcDInPollingService	not passed	EcDInPollingService	One	Created on startup. Deleted on shutdown.
	EcDInProcessingService	DpInPDRParser	Used to parse a PDR file	EcDInProcessingService	not passed	EcDInProcessingService	One per PDR	Created when request is activated. Deleted when parsing has completed.

**Table 4-1. Memory Management Table (3 of 5)**

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/process name) (M)	Passed To (Executable/process name)	Where Deleted? (process name) (M)	Number of Instances (Example – 1 per granule)	Comments/Remarks (Items of special interest. Example - Size per instantiation, never “deleted”, etc.)
	EcDIIInProcessingService	DpInPDR	Used to represent the properties of a request during processing.	EcDIIInProcessingService	not passed	EcDIIInProcessingService	One per request activated	Created when request is activated. Deleted when request processing is complete.
	EcDIIInProcessingService	DpInGranule	Used to represent the properties of a granule during processing.	EcDIIInProcessingService	not passed	EcDIIInProcessingService	One per granule in request	Created when request is activated. Deleted when request processing is complete.
	EcDIIInProcessingService	DpInFile	Used to represent the properties of a file during processing.	EcDIIInProcessingService	not passed	EcDIIInProcessingService	One per file in a granule	Created when granule is activated. Deleted when granule processing is complete.
	EcDIIInProcessingService	DpInProcessingDBInterface	Used to handle all interaction with ingest database	EcDIIInProcessingService	not passed	EcDIIInProcessingService	One	Created on startup. Deleted on shutdown.
	EcDIIInNotificationService	EcDfAutoDispatcher	Email notification queue	EcDIIInNotificationService	not passed	EcDIIInNotificationService	One	Created on startup. Deleted on shutdown.
	EcDIIInNotificationService	EcDfAutoDispatcher	File transfer notification queue	EcDIIInNotificationService	not passed	EcDIIInNotificationService	One	Created on startup. Deleted on shutdown.
	EcDIIInNotificationService	DpInServerMessagesTimer	Used to periodically poll for changes in ingest resource properties.	EcDIIInNotificationService	not passed	EcDIIInNotificationService	One	Created on startup. Deleted on shutdown.
	EcDIIInNotificationService	DpInNotifyPopulateTimer	Used to periodically add new notification actions from ingest database	EcDIIInNotificationService	not passed	EcDIIInNotificationService	One	Created on startup. Deleted on shutdown.

**Table 4-1. Memory Management Table (4 of 5)**

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/process name) (M)	Passed To (Executable/process name)	Where Deleted? (process name) (M)	Number of Instances (Example – 1 per granule)	Comments/Remarks (Items of special interest. Example - Size per instantiation, never “deleted”, etc.)
	EcDllNotificationService	DplnNotifyRemoveCompletedActionsTimer	Used to remove completed notification actions from the ingest database	EcDllNotificationService	not passed	EcDllNotificationService	One	Created on startup. Deleted on shutdown.
	EcDllNotificationService	DplnNotifyEmailAction	Used to perform the notification of ingest via email	EcDllNotificationService	not passed	EcDllNotificationService	One per email notification	Created when new email notification is retrieved from database. Deleted when action has been completed.
	EcDllNotificationService	DplnNotifyFileTransferAction	Used to perform the notification of ingest via file transfer	EcDllNotificationService	not passed	EcDllNotificationService	One per file notification	Created when new file transfer notification is retrieved from database. Deleted when action has been completed.
	EcDllNotificationService	DplnNotifyDatabase	Used to handle all interaction with ingest database	EcDllPollingService	not passed	EcDllPollingService	One	Created on startup. Deleted on shutdown.
	EcDllOdToXml	OdToXmlTranslator	Used to store the ringpoint section of the PDR file in xml format	EcDllOdToXml	not passed	EcDllOdToXml	One	Created on startup. Deleted on shutdown.
	EcDllOdToXml	OdToXmlTranslator	Used to store the PDR file	EcDllOdToXml	not passed	EcDllOdToXml	One	Created on startup. Deleted on shutdown.

**Table 4-1. Memory Management Table (5 of 5)**

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/process name) (M)	Passed To (Executable/process name)	Where Deleted? (process name) (M)	Number of Instances (Example – 1 per granule)	Comments/Remarks (Items of special interest. Example - Size per instantiation, never “deleted”, etc.)
<b>OMS</b>	EcOmOrderManager	OmSrClientDb OmSrDbInterface	Handles connection and queries to the database server.	EcOmOrderManager	Not passed	EcOmOrderManager	One instance	The memory is deallocated when the server comes down.
	EcOmOrderManager	OmSrDispatchQueue	Keeps track of requests for processing.	EcOmOrderManager	Not passed	EcOmOrderManager	Four instances	The memory is deallocated when the server comes down.
	EcOmOrderManager	OmServer	Main encapsulating class.	EcOmOrderManager	Not passed	EcOmOrderManager	One instance	The memory is deallocated when the server comes down.
	EcOmOrderManager	OmSrDistributionRequest	Stores information related to distribution requests.	EcOmOrderManager	Not passed	EcOmOrderManager	One instance per request	The memory is deallocated when the server comes down, or when present request is terminated in any way.
<b>CLS</b>	Not Applicable							
<b>SSS</b>	Not Applicable							
<b>Toolkit</b>	Not Applicable							

## 4.1 Data Server Subsystem Overview

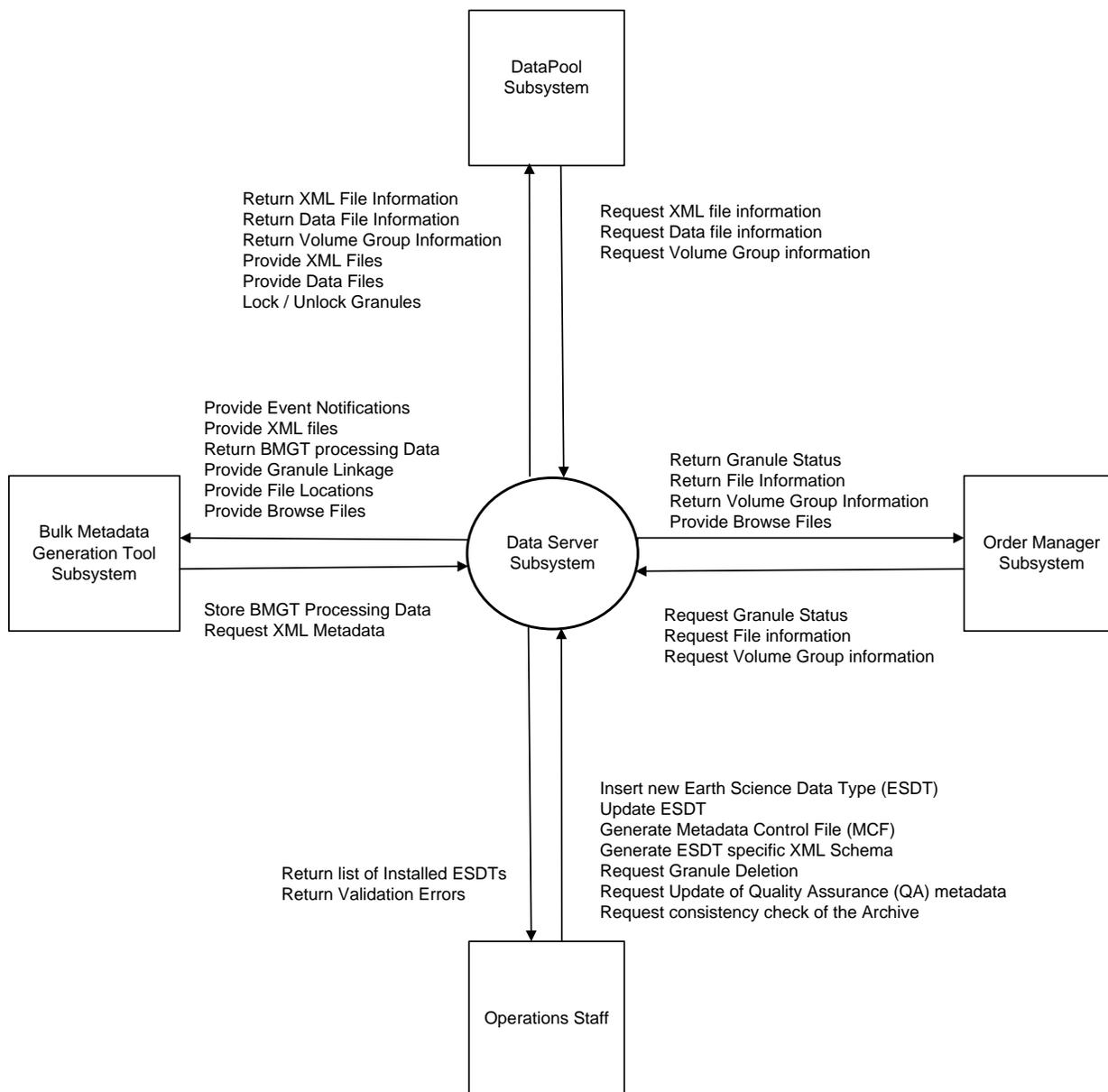
The Data Server Subsystem (DSS) subsystem consists of one CSCI called the “Archive Inventory Management” (AIM) CSCI. All other CSCIs were eliminated in previous ECS releases. The AIM CSCI provides the following services:

- Services for adding new ESDTs
- Services for validation of granule metadata during Ingest
- Recording and tracking the location of files within the Archives
- Recording the addition of new Volume Groups within the Archives
- Supplying events to BMGT
- Providing database functionality for BMGT processing data
- Providing a Universal Reference (UR) for each granule ingested
- Managing the removal of granules from the system
- Managing QA metadata for science granules
- Creation and storage of Metadata Control Files (MCF).

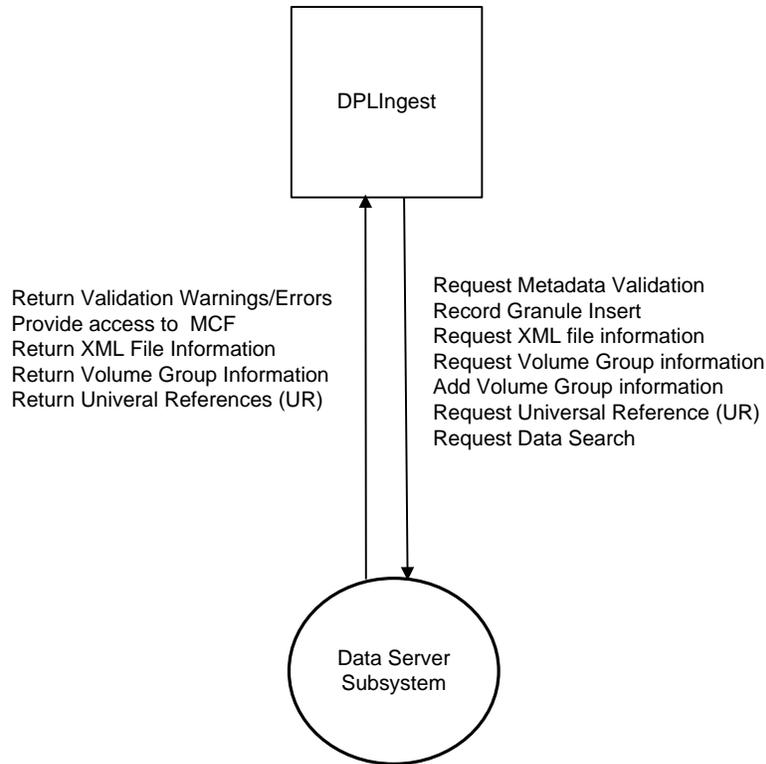
NOTE: As of Release 8.2, the AIM and DataPool Databases have been combined. Since some tables still remain logically a part of the DataPool database despite being physically co-located with the AIM Database, there are still references to the ‘DataPool’ database in this document. Except where otherwise specified, references to the AIM, Inventory, and DataPool databases can be considered to refer to the same physical database.

### Data Server Subsystem Context

Figure 4.1-1 shows context diagrams for the DSS subsystem. These diagrams illustrate the interaction of DSS with other ECS subsystems. DPLIngest was separated out into its own diagram because all the subsystems could not fit into one diagram.



**Figure 4.1-1. Data Server Subsystem Context Diagram (1 of 2)**



**Figure 4.1-1. Data Server Subsystem Context Diagram (2 of 2)**

Table 4.1-1 provides a description for each of the interface events shown in the Data Server Subsystem context diagrams. If the interface is shared between multiple subsystems the interface event will only occur one time in the table and all subsystems that share the interface will be listed in the description.

**Table 4.1-1. Data Server Subsystem Interface Events (1 of 3)**

Event	Interface Event Description
Request XML file information	When the <b>DPL Ingest</b> component ingests granules such as Browse, Processing History, or QA, it requests the location of the XML file for the associated science granule from the <b>DSS Inventory Database</b> and updates the linkage information within that XML file. The <b>DataPool Action Driver (DPAD)</b> component requests XML file information when staging granules into the DataPool.
Request Data File	The <b>DPAD</b> component requests data file information from the <b>DSS Inventory Database</b> when staging granules into the DataPool.
Request Volume Group Information	The <b>DPLIngest</b> component requests volume group information from the <b>DSS AIM Inventory Database</b> to determine where to store the files being ingested. The <b>DPAD</b> component requests volume group information when staging granules into the DataPool. Additionally the <b>OMS</b> and <b>BMGT</b> subsystems requests browse granule location information.
Return XML File Information	The <b>DSS Inventory Database</b> provides XML file name and location information to the <b>DataPool</b> and <b>DPLIngest</b> .
Return Data File information	The <b>DSS Inventory Database</b> provides data file name, size, checksum, and location information to the <b>DataPool</b> .
Return Volume Group Information	The <b>DSS Inventory Database</b> provides a list of directories in the Archive as well as information to determine where each granule is archived to <b>DataPool</b> , <b>OMS</b> , and <b>DPLIngest</b> .
Provide XML Files	The <b>DataPool</b> and <b>BMGT</b> subsystems read XML files directly from the <b>AIM XML Archive</b> .
Provide Data Files	The <b>DataPool</b> reads Science and Browse granule data files directly form the <b>AIM Granule Archive</b> .
Lock / Unlock Granules	The <b>AIM XRU</b> and <b>QA Update utility</b> use the <b>DataPool database</b> to lock and unlock granules while they are being processed so that they cannot be accessed by other ECS components.
Request Granule Status	The <b>OMS</b> subsystem requests the information from the <b>DSS Inventory Database</b> to determine if a granule is “marked for deletion”, or “hidden” and thus can’t be ordered.
Request File Information	The <b>OMS</b> requests file name, size, and checksum information from the <b>DSS Inventory Database</b> while processing orders.
Return Granule Status	The <b>AIM Inventory Database</b> returns information to allow <b>OMS</b> to determine if a granule can be ordered.
Return File Information	The <b>AIM Inventory database</b> provides information such as file size, checksum, internal and distribution file names to the <b>OMS</b> .
Provide Browse Files	The <b>OMS</b> and <b>BMGT</b> subsystems read browse files directly from the <b>AIM Granule Archive</b> .
Insert New Earth Science Data Type	The <b>DSS ESDT Maintenance GUI</b> allows <b>operators</b> to add ESDTs to the system by providing a descriptor file for the ESDT.
Update ESDT	The <b>DSS ESDT Maintenance GUI</b> allows <b>operators</b> to modify a limited set of attributes associated with an installed ESDT by providing a replacement descriptor file for the ESDT.

**Table 4.1-1. Data Server Subsystem Interface Events (2 of 3)**

Event	Interface Event Description
Generate Metadata Control File	The <b>DSS ESDT Maintenance GUI</b> allows <b>operators</b> to create an MCF file for each ESDT added, it also contains a function to generate new MCFs for each installed ESDT.
Request Granule Deletion	The <b>DSS Granule Deletion utilities</b> provides the <b>operator</b> the ability to identify granules to be deleted, mark them for future deletion, return them from a “marked for deletion” to an active status, and physically remove (delete) them from the archives and Inventory database.
Request Update of Quality Assurance metadata	The <b>DSS Quality Assurance Update Utility (QAUU)</b> provides the <b>operator</b> the ability to modify/add metadata related to the quality of the granule to the XML file associated with the granule.
Request consistency check of the Archive	The <b>DSS Archive Check utility</b> allows the <b>operator</b> to check the files stored in the archives against the records stored in the Inventory database.
Return list of Installed ESDTS	The <b>DSS AIM ESDT Maintenance GUI</b> provides <b>operator</b> with a list of ESDTs installed within the Inventory.
Return Validation Errors	The <b>DSS AIM ESDT Maintenance GUI</b> , when inserting or updating an ESDT, reports to the <b>operator</b> any errors found when processing the Descriptor.
Store BMGT Processing Data	The <b>DSS Inventory Database</b> provides a storage location for <b>BMGT</b> processing data, including the events to be processed.
Request XML Metadata	The <b>DSS Inventory Database</b> provides the location of XML files within the DSS XML Archive.
Provide Event Notifications	The <b>DSS Inventory Database</b> records real-time events for <b>BMGT</b> to process.
Provide XML files	The <b>BMGT</b> reads XML metadata from the <b>DSS XML Archive</b> (both descriptors and granule XML files) when processing collections and granules. The descriptors are stored in ODL format and are translated to XML for BMGT.
Provide Granule Linkage	The <b>DSS Inventory Database</b> is used by <b>BMGT</b> to determine the Browse granules that are associated with the Science Granules being processed.
Provide File Locations	The <b>AIM Inventory Database</b> provides the location of XML metadata files and Browse files to the <b>BMGT</b> .
Request XML Metadata Validation	The <b>DPLIngest</b> component uses the <b>DSS XML Validation Utility (XVU)</b> to validate the XML metadata file associated with each granule ingested.
Record Granule Insert	The <b>DPLIngest</b> component records critical metadata about each granule ingested into <b>AIM Inventory Database</b> . This is done via Data Pool Insert Utility (NDPIU), and the metadata is passed to the NDPIU via an XML file. Note: DPLIngest will copy the XML file to the AIM XML Archive.
Add Volume Group Information	The <b>DPLIngest GUI</b> component uses the <b>DSS Inventory database</b> to store information about new volume groups within the archive.
Request Universal Reference	The <b>DSS Inventory Database</b> provides a UR to the <b>DPL Ingest</b> component for each granule ingested.
Request Data Search	The <b>DPLIngest</b> sends a search request to the <b>DSS Inventory Database</b> for a granule corresponding to a particular ESDT short name and version, which has a particular local granule id.

**Table 4.1-1. Data Server Subsystem Interface Events (3 of 3)**

Event	Interface Event Description
Return Validation Warnings/Errors	The <b>DSS XVU</b> returns a list of warning and/or error messages to <b>DPL Ingest</b> if problems were found during the validation of the granule metadata.
Provide access to MCF	The <b>DPLIngest</b> component retrieves the appropriate MCF for the ESDT from the <b>DSS XML Archive</b> when ingesting a non-SIPS granule.
Return Universal Reference	The <b>DSS Inventory Database</b> provides a UR to <b>DPLIngest</b> for each granule ingested.

### Data Server Subsystem Structure

The DSS is one CSCI and two HWCIs:

- The Archive Inventory Management (AIM) CSCI catalogs earth science data as logical collections. Each of these collections is referred to as an “Earth Science Data Type” (ESDT) and the members of the collection are referred to as “Granules.” The AIM CSCI services requests to add new collections to the inventory, add individual granules to existing collections, update granules within a collection, check the consistency of the Inventory database and the Archive file systems, and remove individual granules or whole collections of granules.
- The XML Archive HWCI stores XML files for each science granule in the Inventory as well as descriptor files, MCFs, and XML schema files used for validating XML.
- The Granule Data Archive HWCI provides high-capacity system for the long-term storage of data files.

Detailed information on hardware/software mapping, hardware diagrams, disk partitioning, etc., can be found in 920-TDx-00x, the 921-TDx-00x, and the 922-TDx-00x series of baseline documents. These documents are located at the web site <http://pete.hitc.com/baseline/index.html> and click on the Technical Documents button.

### Use of COTS in the Data Server Subsystem

- RogueWave’s Tools.h++  
The Tools.h++ class libraries provide libraries of object strings and collections. These class libraries are statically linked and delivered with the custom code installation. This library is only used by the Archive consistency checking utility.
- Rogue Wave’s Net.h++  
ToolsPro.h++ is a C++ class library, which includes the net.h++ class library, which provides an object-oriented interface to Inter-Process Communication (IPC) and network communication services. The Net.h++ framework enables developed code to be portable

to multiple operating systems and network services. This library is only used by the Archive consistency checking utility.

- PostgreSQL Server

The PostgreSQL Server provides the capabilities to retrieve, query, insert, update, and delete database records.

- PostgreSQL Client Access Library libpq

The PostgreSQL Client Access Library libpq provides access between DSS custom code and the PostgreSQL backend server.

### **Use of shareware products:**

- XERCES

A library of software for parsing and manipulating XML, it provides the XVU and the ESDT Maintenance GUI the ability to process XML files.

- JDBC

The XVU and the ESDT Maintenance GUI use JDBC Driver to access the PostgreSQL Database Server.

- Standard Java runtime environment

The XVU and the ESDT Maintenance GUI use the standard Java runtime environment to execute their java processes and to provide a set of supporting functionality.

- Java Server Faces

JavaServer Faces provides a web based application framework and user interface components for the ESDT Maintenance GUI. This includes handling navigation between pages, user input, and display.

### **4.1.1 Archive Inventory Management Software Description**

The Archive Inventory Management (AIM) CSCI is composed of several software components. Some of these components were moved to the AIM CSCI from other DSS CSCIs while others are new components used to replace SDSRV functionality. The existing components that are moved to AIM are:

- Granule Deletion utilities

- EcDsBulkSearch
- EcDsBulkDelete
- EcDsBulkUndelete
- EcDsDeletionCleanup

- Quality Assurance Update utility
- Archive Check Utility

The new software components included in the AIM CSCI are:

- ESDT Maintenance GUI
- XML Validation Utility
- XML Replacement Utility
- XML Archive Check Utility

The AIM CSCI is used by other ECS CSCIs to manage the XML metadata and storage location for each granule within the ECS inventory. Table 4.1-2 lists the components of the AIM CSCI along with a brief description of the component.

**Table 4.1-2. AIM Software Components (1 of 3)**

<b>Process</b>	<b>Type</b>	<b>Hardware CI</b>	<b>COTS / Developed</b>	<b>Functionality</b>
EcDsBulkSearch.pl	Command line utility	OMLHW	Developed	EcDsBulkSearch.pl utility provides a command line operator interface for creating a list of granules to be used for Bulk Delete or Bulk Undelete operations.
EcDsBulkDelete.pl	Command Line Utility	OMLHW	Developed	The EcDsBulkDelete.pl utility provides a command line operator interface for deleting granules (marking the granule so that it can't be accessed and making it eligible for future removal) in the Inventory database.
EcDsBulkUndelete.pl	Command line utility	OMLHW	Developed	The EcDsBulkUndelete.pl provides a command line operator interface for changing granules that were previously marked as deleted to an "active" state in the Inventory database.

**Table 4.1-2. AIM Software Components (2 of 3)**

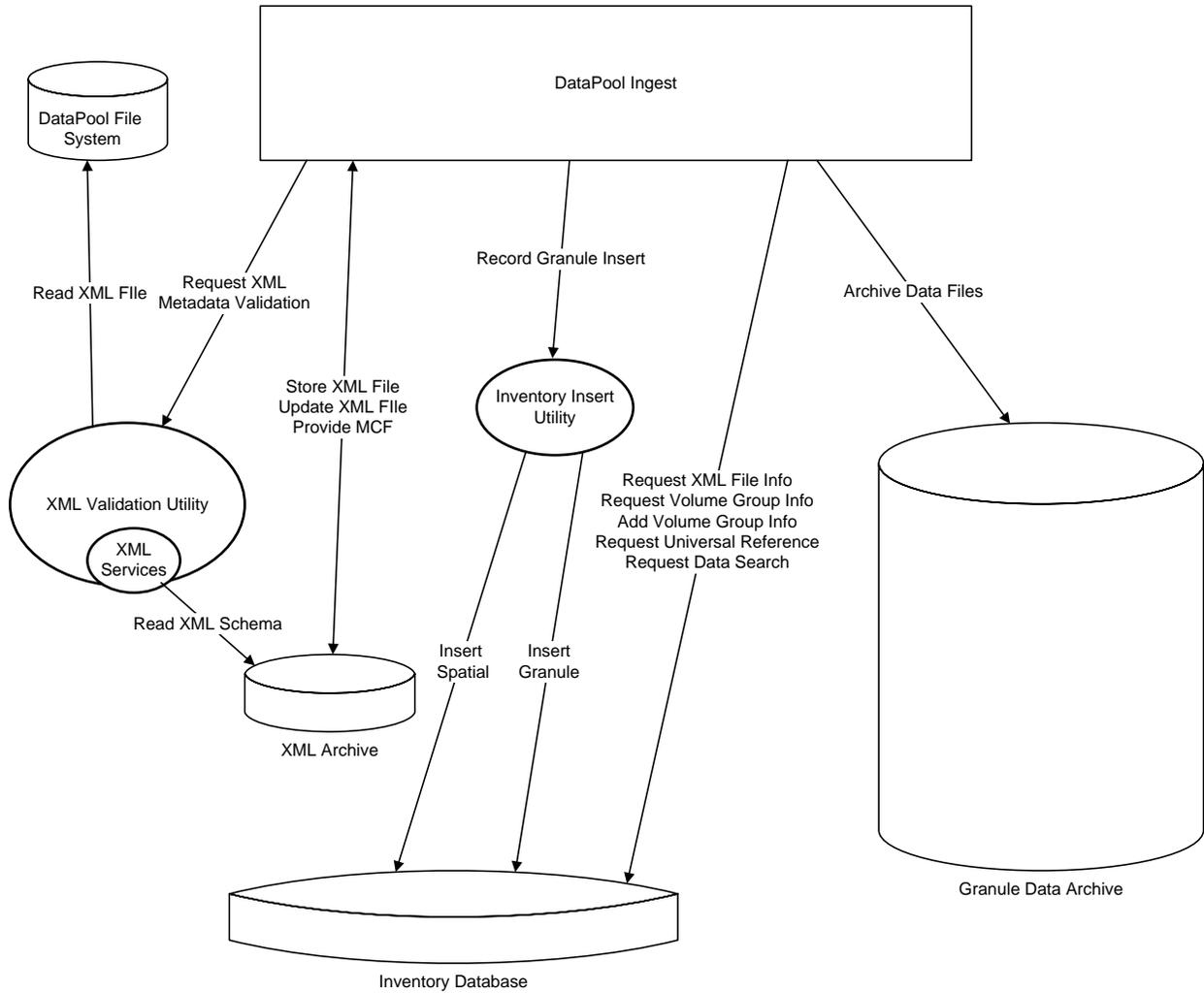
Process	Type	Hardware CI	COTS / Developed	Functionality
EcDsDeletionCleanup.pl	Command line utility	OMLHW	Developed	The EcDsDeletionCleanup.pl provides a command line operator interface for removing the metadata and data files associated with granules that were previously marked for deletion. This utility removes all references to the deleted granules from the AIM CSCI.
EcAmQAUpdateUtility	Command line utility	OMLHW	Developed	The Quality Assurance Update Utility (QAUU) is a java command line utility that updates QA information in the XML Archive and possibly the XML files within the public DataPool. It processes an input file specifying the QA attributes that are being changed along with a specification for the granules to be updated.
EcDsAmArchiveCheckUtility	Command line utility	DPLHW	Developed	The Archive Check is a C++ utility that compares the records of the AIM Inventory database to files in the XML Archive and Granule Archive. It reports discrepancies for both missing files and missing database records.
ESDT Maintenance GUI	GUI	DPLHW	Developed	The ESDT Maintenance GUI is a web based java application that installs, updates, and deletes ESDTs from the system. It can also be used for informational purposes to see which ESDTs are installed in the system and to view an individual descriptor. The GUI is also responsible for generating ESDT specific XML schema files and MCFs from the descriptor and storing them in the XML Archive.
EcDsAmXvu	Utility	DPLHW	Developed	The XML Validation Utility (XVU) is a java process that parses the XML metadata file being ingested and validates it against a set of rules stored in an ESDT specific schema (located in the XML archive).

**Table 4.1-2. AIM Software Components (3 of 3)**

Process	Type	Hardware CI	COTS / Developed	Functionality
EcDsAmXru	Command line utility	DPLHW	Developed	The XML Replacement utility (XRU) is command line java utility that replaces XML files within the XML Archive and creates an event for the BMGT subsystem to process. The XRU validates the new candidate file prior to replacing the original file in the XML Archive.
EcDsCheckXMLArchive.pl	Command line utility	DPLHW	Developed	The XML Archive Check utility compares the files stored in the XML archive to the entries in the AIM Inventory database. This utility is optimized for processing the XML Archive, while the EcAmQAUpdateUtility is optimized for processing volume groups.
PostgreSQL	Server	DBLHW	COTS	The PostgreSQL is the primary database engine for ECS. It used for storing ESDT and granule metadata in the AIM Inventory database.

#### 4.1.1.1 AIM Interfaces with DataPool Ingest

Figure 4.1-2 shows the components of the AIM CSCI that interface with the DPLIngest CSCI. Oval shapes are used to indicate AIM processes. The diagram shows the interactions between the AIM components as well as the data stores within the AIM CSCI. Refer to the DPLIngest section of the 305 document for a complete description of DPLIngest processing.



**Figure 4.1-2. AIM CSCI Context Diagram (DPLIngest)**

Table 4.1-3 explains each of the events/interfaces depicted above.

**Table 4.1-3. AIM Interfaces with DPLIngest (1 of 4)**

Event	Event Frequency	Interface	Initiated By	Event Description
Request XML Metadata Validation	One per granule ingested	EcDsAmXvu	EcDIIInProcessingService	<p>Each granule ingested into ECS must pass a metadata validation step.</p> <ul style="list-style-type: none"> <li>▪ The DPLIngest EcDIIInProcessingService component passes the name and location of the XML metadata file within the DataPool hidden directory to the AIM XVU process.</li> <li>▪ The XVU validates the file using the XML Services jar file and the ESDT specific schema located in the XML Archive.</li> <li>▪ The XVU processes the XML “post validation information set” to determine the outcome of the XML validation.</li> <li>▪ Optional elements that are invalid are removed from the file.</li> <li>▪ The XVU also performs custom validation on certain elements.</li> <li>▪ The XVU returns a value of 0 or success, 2 for failure, 3 to indicate the metadata passed validation but that some optional elements were removed, and finally a value of 4 indicates the request should be retried at a later time.</li> </ul>
Read XML Schema	Once per granule ingested		EcDsAmXvu	<p>When validating a granule XML file, the XVU reads the ESDT specific XML schema from the XML Archive. The schema is created as part of installing the ESDT.</p>

**Table 4.1-3. AIM Interfaces with DPLIngest (2 of 4)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Read XML File	Once per granule ingested		EcDsAmXvu	The XML file is read into a DOM tree within the XVU process.
Store XML File	Once per science granule ingested		EcDIInProcessingService	The EcDIInProcessingService stores an XML metadata file for each science granule ingested. XML files are not archived for Browse, QA, Production History, or Delivered Algorithm Package granules.
Update XML File	Once per browse granule ingested		EcDIInProcessingService	When the EcDIInProcessingService ingests a Browse granule, it adds the Browse ID to the XML metadata file of each science granule that is referenced by the Browse granule linkage information.
Provide MCF	Once per non-SIPs granule ingested		EcDIInProcessingService	When Ingest needs an MCF to preprocess a Non-SIPS ingest request, it reads the MCF directly from the Small File Archive. The directory where the MCFs are stored is a configuration item in Ingest.
Request XML File Info		Stored Procedure	EcDIInProcessingService	Ingest reads the list of XML paths from the Inventory database. Ingest creates new paths in the XML Archive for each ESDT when it ingests the first granule for the ESDT for the current month. This new directory is recorded in the Inventory database as part of the Record Granule Insert event.

**Table 4.1-3. AIM Interfaces with DPLIngest (3 of 4)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Request VolumeGroup Info	Once at startup of EcDlInProcessingService or after new groups are added.	Stored Procedure	EcDlInProcessingService	Ingest reads the list of open volume groups from the Inventory database at startup.
Add Volume Group	Upon request by DAAC operators	Stored Procedure	DAAC Operator / DPL Ingest GUI	The DAAC operator uses the DPL Ingest GUI to create new volume groups. The GUI inserts these into the Inventory database via stored procedure. When this happens, a message is created in the Ingest database to instruct Ingest to refresh the cached list of volume groups.
Request Universal Reference	Once for each granule Ingested	Stored Procedure	EcDlInProcessingService	Ingest accesses the Inventory database to get the next available granule ID. It then uses its configured UR prefix and the ShortName and VersionID of the granule to construct a Universal Reference.
Request Data Search	Once for each Browse granule Linkage	Stored Procedure	EcDlInProcessingService	Ingest receives Local Granule ID values in the linkage section of the Browse ingest request. It searches the Inventory database (via stored procedure) to convert these to granule ID values.
Record Granule Insert	Once for each granule Ingested	NDPIU	EcDlInProcessingService	Ingest sends a request to the Data Pool Insert Utility to record each granule ingested.
Insert Spatial	Once per science granule with spatial attributes	Embedded SQL	NDPIU	The NDPIU inserts the spatial metadata into the Inventory database via the Spatial Query Server. The spatial metadata is inserted before the non-spatial metadata. Each use separate database transactions. If the non-spatial insert fails, then the spatial is removed. In the event that the spatial fails due to a duplicate key error, the NDPIU assumes it is processing a retry of the metadata insert and proceeds to the non-spatial insert.

**Table 4.1-3. AIM Interfaces with DPLIngest (4 of 4)**

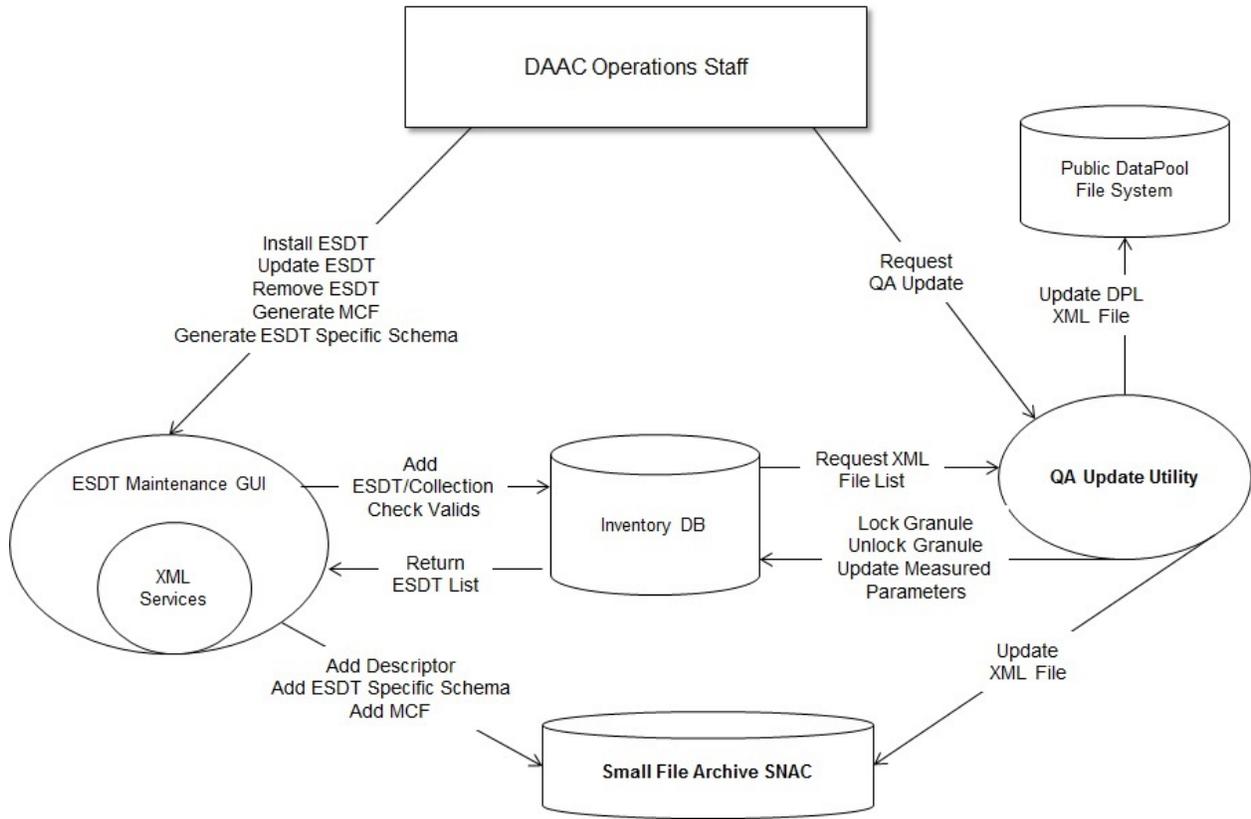
<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Insert Granule	Once per granule ingested	<i>Stored procedure</i>	NDPIU	The NDPIU records non-spatial information such as the temporal metadata, the file information, and the XML file location in the Inventory database.
Archive Data Files	Once per granule ingested		EcDIInProcessingService	Ingest copies the data files for the granule into the data archive. The location/directory is determined by executing an Inventory database stored procedure to compare the granule metadata with the DsStVolumeGroup table.

#### **4.1.1.2 AIM CSCI interfaces with DAAC Operations Staff**

There are several interfaces between the DAAC operations staff and the Archive Inventory Management (AIM) CSCI. These can't be show in one diagram so this section is broken up into several sections, each explaining a common set of DAAC operator interfaces. Each figure is followed by a table that describes the interfaces / events shown in the figure.

##### **4.1.1.2.1 AIM ESDT Maintenance GUI and QA Update utility**

Figure 4.1-3 shows the DAAC operator interfaces with the ESDT Maintenance GUI and the QA Update utility and interaction of these components with other AIM components.



**Figure 4.1-3. AIM Interfaces with DAAC Operators (ESDT Maintenance GUI and QA Update utility)**

Table 4.1-4 describes each of the events depicted above.

**Table 4.1-4. AIM Interfaces with DAAC Operators (ESDT GUI, QA Update) (1 of 5)**

Event	Event Frequency	Interface	Initiated By	Event Description
Install ESDT	Once for each new ESDT	ESDT Maintenance GUI	DAAC Operations	<p>The DAAC operators install ESDTs using the ESDT Maintenance GUI. The GUI performs the following:</p> <ul style="list-style-type: none"> <li>• Reads the descriptor files from a configured directory.</li> <li>• Converts the descriptor to XML.</li> <li>• Records the ESDT in the Inventory database.</li> <li>• Validates the descriptor against an ESDT XML schema using the XML services module.</li> <li>• Validates certain elements of the descriptor against valids stored in the Inventory database.</li> <li>• Adds the Collection entry to the Inventory database.</li> <li>• Registers an Insert Event in the Spatial Subscription Server database.</li> <li>• Extracts an MCF from the descriptor and stores it in the XML archive.</li> <li>• Builds an ESDT specific XML schema file and stores it in the XML Archive.</li> <li>• Records the completion of the install in the Inventory database.</li> </ul>

**Table 4.1-4. AIM Interfaces with DAAC Operators (ESDT GUI, QA Update) (2 of 5)**

Event	Event Frequency	Interface	Initiated By	Event Description
Update ESDT	When ever an ESDT descriptor file changes	ESDT Maintenance GUI	DAAC Operations	<p>When an ESDT is updated the ESDT Maintenance GUI performs the following:</p> <ul style="list-style-type: none"> <li>• Reads the new descriptor file from the configured directory.</li> <li>• Converts the descriptor to XML.</li> <li>• Records the update of the ESDT in the Inventory database.</li> <li>• Validates the descriptor against an ESDT XML schema using the XML services module.</li> <li>• Updates the Collection entry in the Inventory database. Note: changes to metadata such as spatial search type, Product Specific Attribute definitions, and other attributes that could invalidate existing metadata are not supported.</li> <li>• Extracts the MCF the from the descriptor and stores it in the XML archive</li> <li>• Builds an ESDT specific XML schema file and stores it in the XML Archive.</li> <li>• Records the completion of the update in the Inventory database.</li> </ul>

**Table 4.1-4. AIM Interfaces with DAAC Operators (ESDT GUI, QA Update) (3 of 5)**

Event	Event Frequency	Interface	Initiated By	Event Description
Remove ESDT	Upon Operator request (after all granules are removed)	ESDT Maintenance GUI	DAAC Operations	<p>ESDTs can be removed from the system if all the granules are physically deleted. The ESDT Maintenance GUI performs the following:</p> <ul style="list-style-type: none"> <li>• Marks the ESDT as being removed in the Inventory database</li> <li>• Removes the MCF and XML schema from the XML archive</li> <li>• Removes the Insert Event from the SSS database</li> <li>• Removes any remaining XML metadata file directories from the XML archive</li> <li>• Removes the collection from the Inventory database</li> <li>• Removes the ESDT information from the Inventory database.</li> </ul>
Generate MCF	Once per ESDT Install Update or upon direct request by the operator		ESDT Maintenance GUI	The ESDT Maintenance GUI extracts the MCF section from the descriptor file and stores it as a separate file in a configured MCF directory within the XML archive.
Generate ESDT specific schema	Once per ESDT Install or Update or upon direct request by the operator		ESDT Maintenance GUI	The ESDT Maintenance GUI reads the Inventory section of the descriptor file and compares it to a "common" granule XML schema. It adds the common schema elements that match the descriptor entries to the new ESDT specific XML schema file and stores it in the configured descriptor directory of the XML Archive. The rules in the descriptor can be set to customize the element definitions (mandatory/optional or specific domain list) in the ESDT specific schema.
Add ESDT / Collection	Once per ESDT Install	Stored procedure	ESDT Maintenance GUI	The ESDT Maintenance GUI executes several stored procedures to register the ESDT in the Inventory database.

**Table 4.1-4. AIM Interfaces with DAAC Operators (ESDT GUI, QA Update) (4 of 5)**

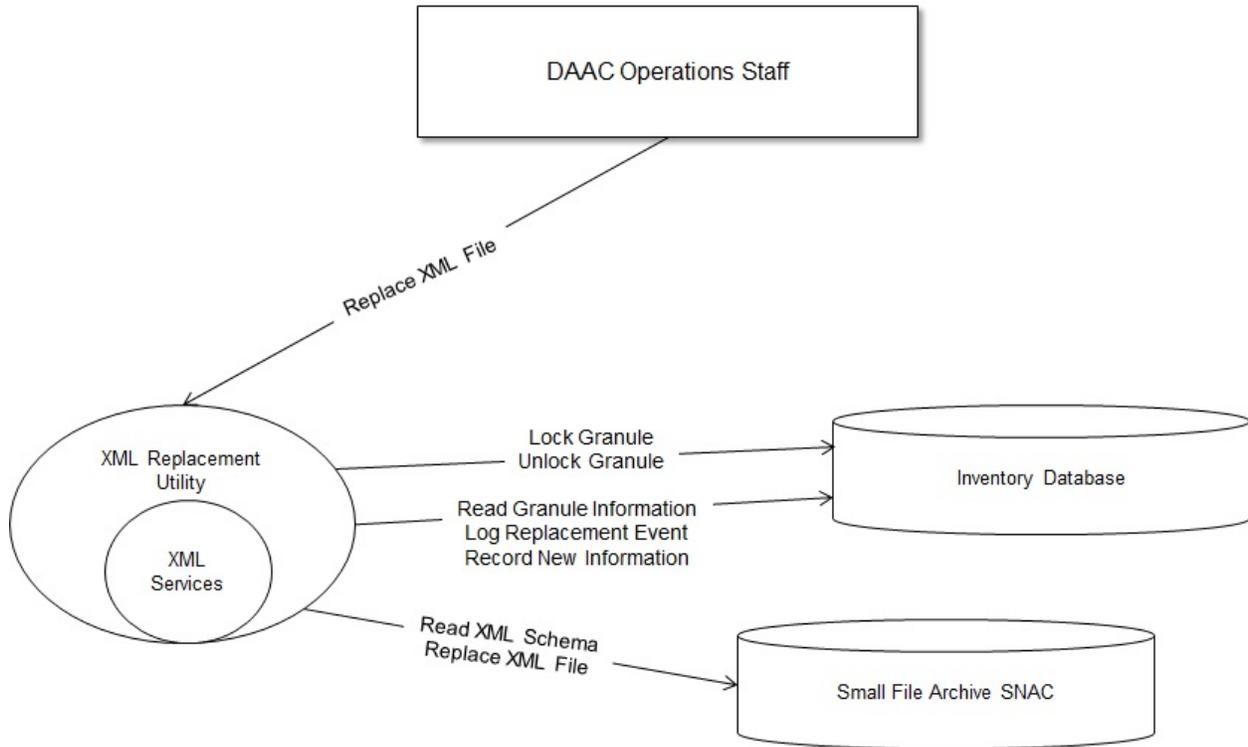
Event	Event Frequency	Interface	Initiated By	Event Description
Check Valid	Once per ESDT Install	Stored procedure	ESDT Maintenance GUI	The ESDT Maintenance GUI checks certain values in the descriptor file (for example: Discipline, Topic, Term Keyword) against a set of valid values stored in the Inventory database.
Return ESDT List	Upon request by Operator	Stored procedure	ESDT Maintenance GUI	The ESDT Maintenance GUI retrieves the list of installed ESDTs and presents them to the Operator.
Add Descriptor	Once per ESDT Install or Update		ESDT Maintenance GUI	The ESDT Maintenance GUI copies or replaces the descriptor file to the configured descriptor directory within the XML archive.
Add ESDT specific schema	Once per ESDT Install or Update		ESDT Maintenance GUI	The ESDT Maintenance GUI stores the ESDT specific granule XML schema in the descriptor directory of the XML archive. This schema is used to validate granules during Ingest.
Add MCF	Once per ESDT Install or Update		ESDT Maintenance GUI	The ESDT Maintenance GUI extracts the Inventory section of the descriptor file and stores it in the configured MCF directory within the XML archive. This file can be accessed directly by DPL Ingest.
Request QA Update	Whenever quality information for a granule or set of granules is available	EcDsAmQauu	DAAC Operations	The DAAC operator submits a file that specifies the granules to be updated along with the new values for the Quality Flags for a Measured Parameter. The QAUU processes the file by uploading it to one of three different request tables (based upon the file format LGID, dbID, or ESDT + Temporal). The request details are copied to a new table where the information is normalized into a set of granules including dbID and Local Granule ID. The QAUU processes the list of granules to be updated in batches.

**Table 4.1-4. AIM Interfaces with DAAC Operators (ESDT GUI, QA Update) (5 of 5)**

Event	Event Frequency	Interface	Initiated By	Event Description
Lock Granule	Executed once per batch, locking each granule within the batch	Stored Procedure	EcDsAmQauu	The QAUU coordinates with other processes that access granules in the DataPool by locking the granule in the AIM database (inserting to the DIOMSGranules table). All the granules within a batch are locked prior to processing the batch.
Request XML File List	Once per batch of granules processed	Stored Procedure	EcDsAmQauu	The QAUU requests the list of XML files that correspond to the granules being processed.
Update XML File	Once per granule processed		EcDsAmQauu	Depending upon the header file line, the QAUU either updates the ScienceQualityFlag or the OperationalQualityFlag, or the AutomaticQuality flag for the supplied Measured Parameter for each granule. It also updates the associated QualityFlagExplanation.
Update DPL XML File	Once per public DPL granule processed		EcDsAmQauu	For each granule within the batch to process, If the granule is in the public DataPool, the QAUU copies the updated XML file from the XML Archive to the DPL file system.
Update Measured Parameters	Once per batch of granules to process	Stored Procedure	EcDsAmQauu	The QAUU updates the quality flags within the AIM database for each supplied Measured Parameter for each of the granules in the batch. This update occurs as one large database transaction.
Unlock Granule	Once per batch of granules to process	Stored Procedure	EcDsAmQauu	When the QAUU finishes processing a batch of granules, it unlocks them in the AIM database by removing the entries from the DIOMSGranules table.

#### 4.1.1.2.2 AIM XML Replacement and Granule Deletion utilities

Figure 4.1-4 shows the interactions between the XML Replacement Utility and the other AIM components. There is only one use case for the XRU and it is initiated by the DAAC operator. The diagram also shows an interface to the DataPool database.



**Figure 4.1-4. AIM Interfaces with DAAC Operators (XML Replacement Utility)**

Table 4.1-5 describes the interfaces / events depicted above between the DAAC operators and the Granule Deletion utilities.

**Table 4.1-5. AIM Interfaces with DAAC Operators (XML Replacement Utility)  
(1 of 2)**

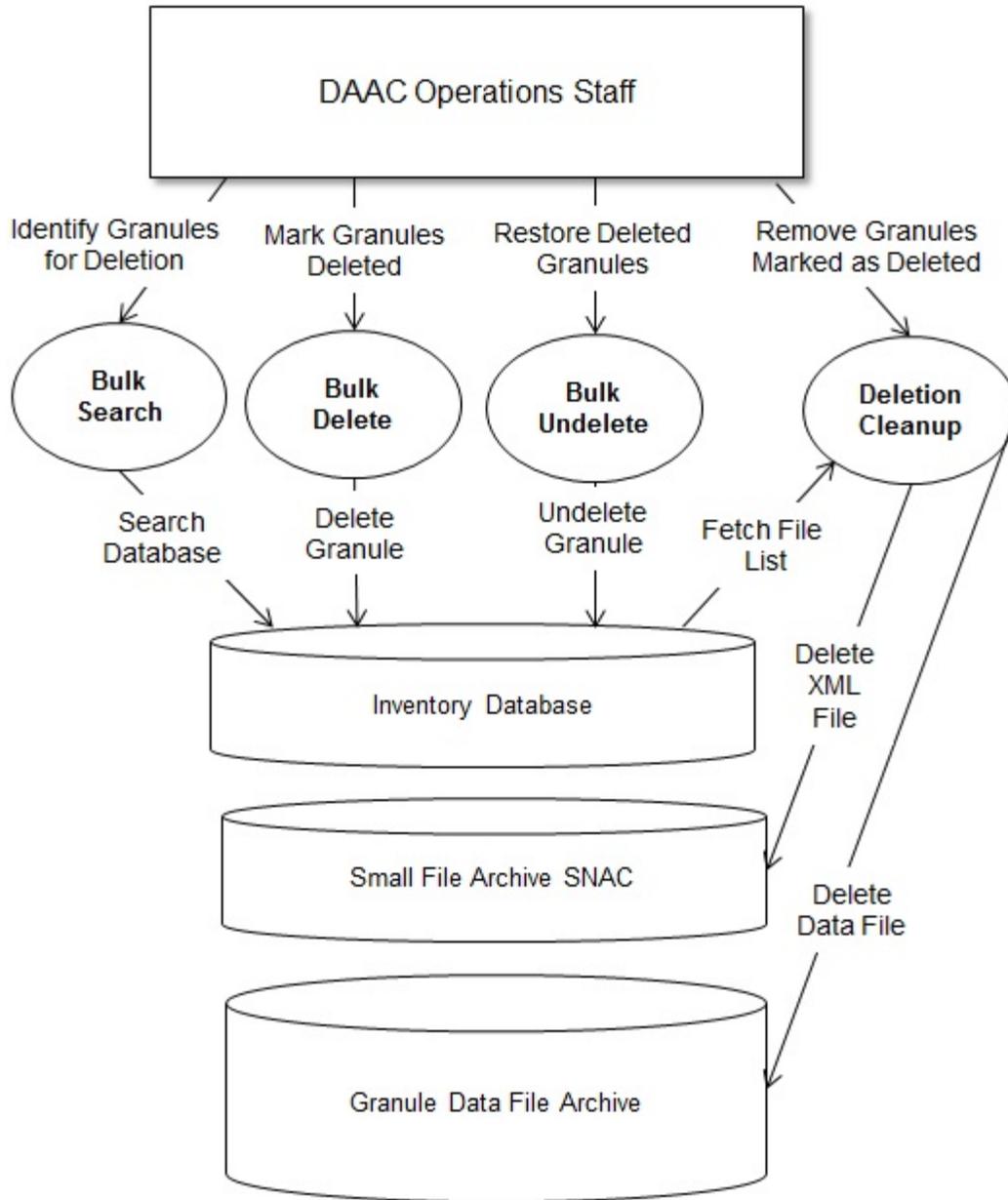
Event	Event Frequency	Interface	Initiated By	Event Description
Replace XML File	Upon request by the Operator	<i>EcDsAmXru</i>	<i>DAAC Operator</i>	The XML Replacement utility replaces XML files in the XML archive and logs an event for BMGT to process.
Read Granule Information	Once per XML file to replace	<i>Stored Procedure</i>	<i>EcDsAmXru</i>	The XRU reads the last update time of the Granule from the Inventory database and compares it to the time in the XML file. It also reads the location of the XML file in the XML archive.

**Table 4.1-5. AIM Interfaces with DAAC Operators (XML Replacement Utility)  
(2 of 2)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Lock Granule	Once per XML file to replace	<i>Stored Procedure</i>	<i>EcDsAmXru</i>	The XRU locks the granule in the DataPool so that it cannot be accessed by other ECS processes while it is being replaced.
Read XML Schema	Once per XML file to replace		<i>EcDsAmXru</i>	The XRU reads the XML file into a DOM tree so that the elements can be validated.
Replace XML File	Once per XML file to replace		<i>EcDsAmXru</i>	If the elements of the new XML file are valid, the existing file in the XML Archive is replaced with the new file.
Record new Information	Once per XML file to replace	<i>Stored Procedure</i>	<i>EcDsAmXru</i>	The XRU records a new size, checksum, and lastUpdate time in the Inventory database for the granule.
Log Replacement Event	Once per XML file to replace	<i>Stored Procedure</i>	<i>EcDsAmXru</i>	The XRU records an event in the Inventory database to be processed by BMGT.
Unlock Granule	Once per XML file to replace	<i>Stored Procedure</i>	<i>EcDsAmXru</i>	The XRU removes the lock from the DataPool database once processing is completed.

#### **4.1.1.3 AIM Granule Deletion utilities**

Figure 4.1-5 shows the DAAC operator interfaces between the DAAC operator and the XML Replacement and Granule Deletion utilities.



**Figure 4.1-5. AIM Interfaces with DAAC Operators (Granule Deletion Utilities)**

Table 4.1-6 describes the interfaces / events depicted above between the DAAC operators and the Granule Deletion utilities.

**Table 4.1-6. AIM Interfaces with DAAC Operators (Granule Deletion) (1 of 2)**

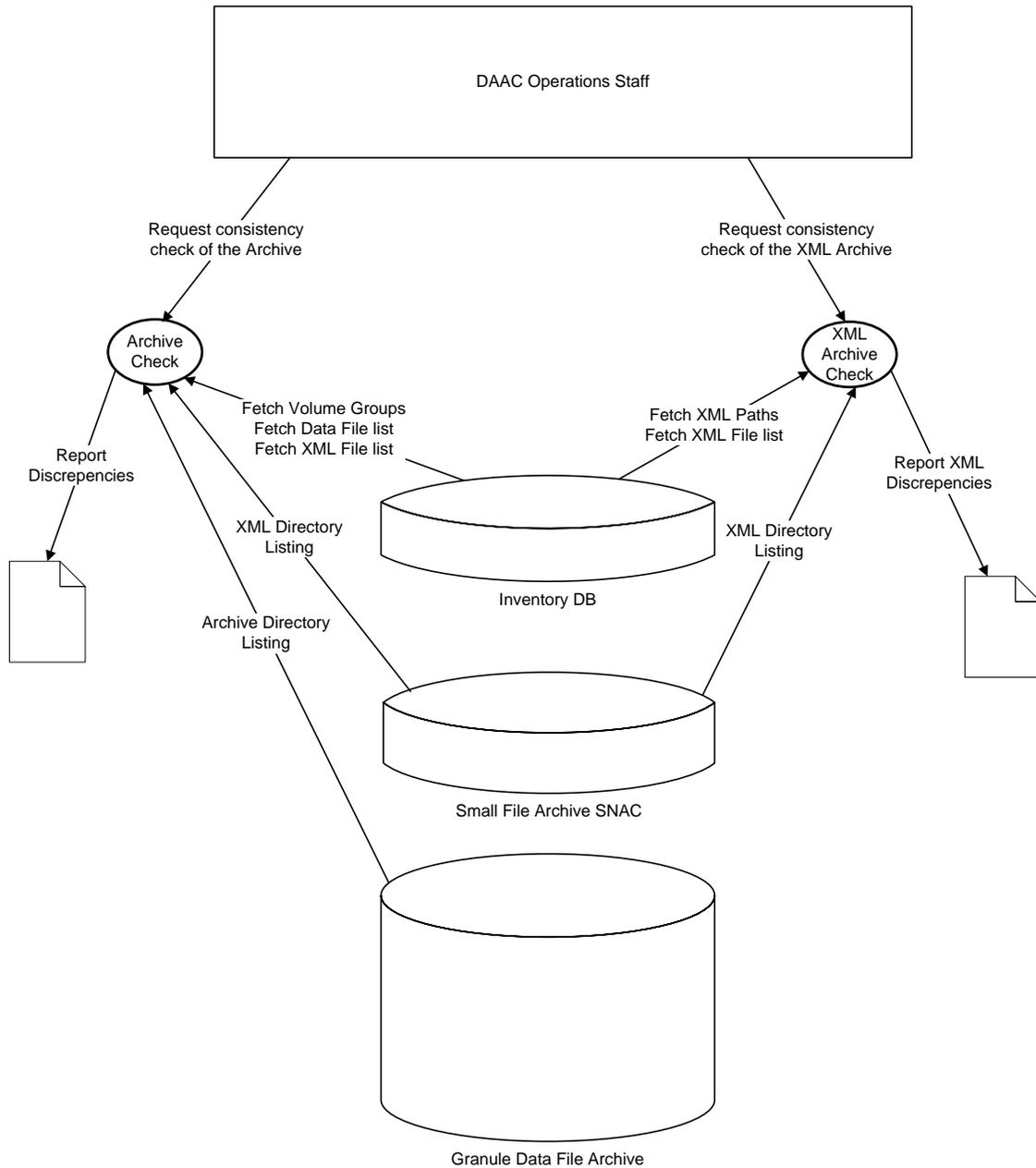
<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Identify Granules for Deletion	Upon request by the Operator	<i>EcDsBulkSearch.pl</i>	<i>DAAC Operator</i>	The Bulk Search utility searches the Inventory database based upon the supplied arguments to identify a list of granules. The list is stored in a text file that is compatible with both the Bulk Delete and the Bulk Undelete utilities.
Search Database	Once per invocation of the utility	<i>Dynamic SQL</i>	<i>EcDsBulkSearch.pl</i>	The Bulk Search utility dynamically constructs an SQL statement based upon the arguments given. These can include ShortName, VersionID, Temporal Range, insert time, Local Granule ID, and current deletion status. The output of the search is a list of Granule IDs stored in a text file.
Mark Granules Deleted	Upon request by DAAC Operator	<i>EcDsBulkDelete.pl</i>	<i>DAAC Operator</i>	The Bulk Delete utility processes a file containing a list of Granule IDs and marks each granule as either "deleted from the Archive" or marks each granule as deleted on the current date. Options exist for marking associated granules. The associated granules (Browse, QA, or PH) can be ignored or they can also be marked as deleted at the same time the science granule is deleted.
Delete Granule	Once per granule deleted	<i>Stored Procedure</i>	<i>EcDsBulkDelete.pl</i>	The Bulk Delete utility executes a stored procedure to update the granule in the Inventory database with the appropriate deletion status.
Bulk Undelete	Upon request by DAAC Operator	<i>EcDsBulkUndelete.pl</i>	<i>DAAC Operator</i>	The Bulk Undelete processes a file of Granule IDs and either reverses the Delete From Archive or the Granule Deletion status (based upon the arguments supplied). The utility also has options for processing associated granules.

**Table 4.1-6. AIM Interfaces with DAAC Operators (Granule Deletion) (2 of 2)**

Event	Event Frequency	Interface	Initiated By	Event Description
Undelete Granule	Once per granule processed	<i>Stored Procedure</i>	<i>EcDsBulkUndelete.pl</i>	The Bulk Undelete utility executes a stored procedure to reverse the deletion status for each granule supplied in the input file. Associated granules are also processed after all science granules are individually processed. This is done in a single stored procedure based upon the time a transaction time for the undelete operation.
Remove Granules marked as deleted	Upon request by DAAC operator	<i>EcDsDeletionCleanup.pl</i>	<i>DAAC Operator</i>	The Deletion Cleanup utility examines the Inventory database for granules that are eligible for removal. The eligible granules have a delete effective date in the Inventory database (set by the Bulk Delete utility) that is less than the time argument passed into the utility. This time argument is referred to as the "lag time" for deleting granules. Once the lag time has passed, the granule is eligible for physical removal from the system.
Fetch File list	Once per invocation of the utility	<i>Stored Procedure</i>	<i>EcDsDeletionCleanup.pl</i>	The Deletion Cleanup utility uses a stored procedure in the Inventory database to expand the list of granules to delete to cover both the primary and backup volume groups and return the actual list of files.
Delete XML File	Once per file processed		<i>EcDsDeletionCleanup.pl</i>	For each file to delete, the Deletion Cleanup utility removes the file from the XML Archive.
Delete Data File	Once per file processed		<i>EcDsDeletionCleanup.pl</i>	For each data file to delete, the Deletion Cleanup utility removes the file from both the primary and backup (if applicable) volume group(s).

### 4.1.1.3.1 AIM Archive Check and XML Archive Check utilities

Figure 4.1-6 shows the interactions between the Archive Check and XML Archive Check utilities and other AIM components. Both utilities are initiated by the DAAC operator.



**Figure 4.1-6. AIM Interfaces with DAAC Operators (Archive Check Utilities)**

Table 4.1-7 describes the interfaces / events depicted above.

**Table 4.1-7. AIM Interfaces with DAAC Operators (Archive Check Utilities) (1 of 3)**

Event	Event Frequency	Interface	Initiated By	Event Description
Request consistency check of the Archive	Upon request by Operations	<i>EcDsAmArchiveCheckUtility</i>	<i>DAAC Operations</i>	<p>The Archive Check utility obtains a list of Volume Groups from the Inventory database and for each Volume Group, the utility compares the list of files in the Volume Group to the entries recorded in the Inventory database that correspond to granules that are mapped to the Volume Group. The utility also checks to make sure every science granule within the Volume Group has an XML file in the appropriate directory of the XML Archive.</p> <p>The mapping of data files to granules is based upon ShortName, VersionID, insert time into the archive, and a comparison of the acquisition time of the granule with a date within the Volume Group to indicate a forward processing or reprocessing use. This rule can be executed by any process to determine the location of the data file.</p> <p>XML Files are stored based upon the Year and Month of the acquisition time of the granule, or the Year and Month of the insert time, if acquisition time is not captured. The Inventory database directly records this association to the absolute directory where the XML file is stored.</p>
Request consistency check of the XML Archive	Upon request by Operations	<i>EcDsAmXMLArchiveCheck</i>	<i>DAAC Operations</i>	<p>The XML Archive Check utility iteratively processes each XML metadata directory and compares the contents of the directory with the contents of the Inventory database. It begins with getting a list of XML metadata directories from the database. Then loops through each one doing the consistency check.</p>

**Table 4.1-7. AIM Interfaces with DAAC Operators (Archive Check Utilities) (2 of 3)**

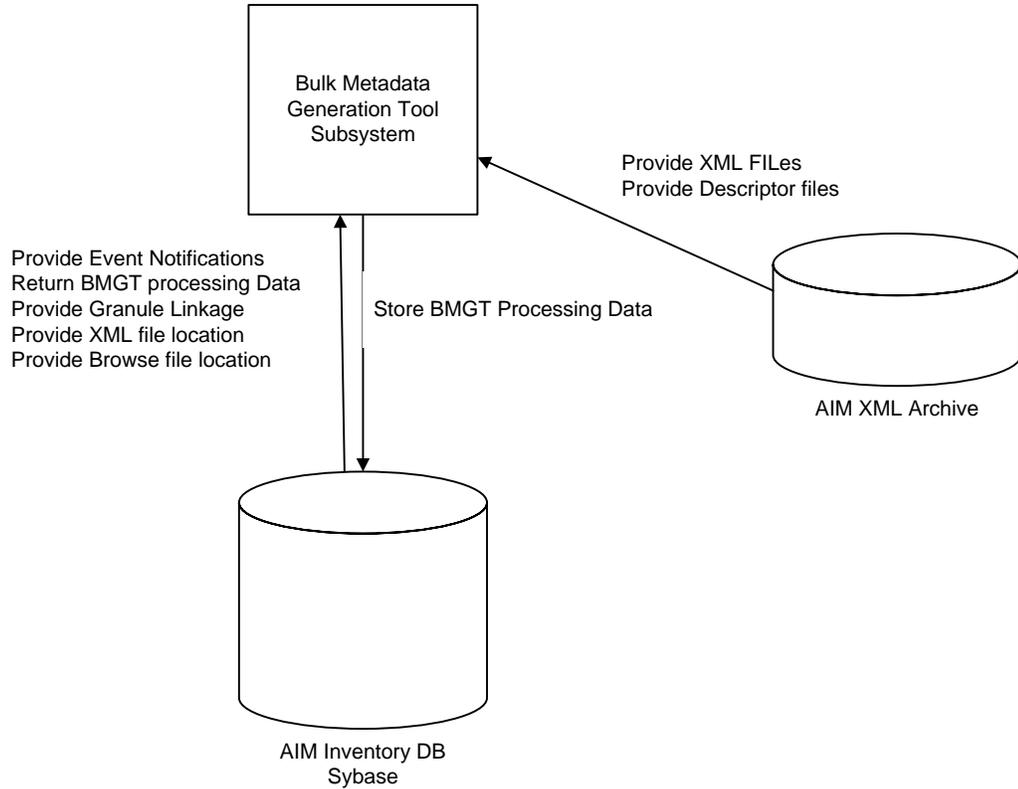
Event	Event Frequency	Interface	Initiated By	Event Description
Fetch Volume Groups	Once per invocation of the utility	<i>Stored Procedure</i>	<i>EcDsAmArchiveCheckUtility</i>	The Archive Check utility retrieves the list of Volume Groups to process from the Inventory database.
Fetch Data File list	Once per Volume Group processed	<i>Stored Procedure</i>	<i>EcDsAmArchiveCheckUtility</i>	The Archive Check utility retrieves the "internal" file name associated with each granule that is mapped to the Volume Group being processed. The mapping is based upon ESDT, insert time into the archive, and a comparison of the acquisition time of the granule with a date within the Volume Group to indicate a forward processing or reprocessing use.
Fetch XML File list	Once per volume group processed  Once per XML metadata directory	<i>Stored Procedure</i>	<i>EcDsAmArchiveCheckUtility</i>  <i>EcDsAmCheckXMLArchive</i>	The Archive Check utility retrieves a list of XML files associated with each granule to be processed within the Volume Group. The XML Archive Check utility iteratively retrieves the list of XML files in each XML directory.
XML Directory listing	Once per granule processed		<i>EcDsAmArchiveCheckUtility</i>  <i>EcDsAmCheckXMLArchive</i>	The Archive Check utility verifies that each granule being processed has an XML file in the XML Archive.  The XML Archive Check utility obtains a list of all files within the XML directory being processed. This will be compared to the contents of the Inventory database which was determined in "Fetch XML File list."
Archive Directory listing	Once per Volume Group processed		<i>EcDsAmArchiveCheckUtility</i>	The Archive Check utility obtains a list of files in the Volume Group being processed and compares it to the list of internal file names retrieved from the Inventory database.

**Table 4.1-7. AIM Interfaces with DAAC Operators (Archive Check Utilities) (3 of 3)**

Event	Event Frequency	Interface	Initiated By	Event Description
Report Discrepancies	Once per invocation of the utility		<i>EcDsAmArchiveCheckUtility</i>	The Archive Check utility writes a discrepancy report to a text file. The report lists granules without data files or XML files in the file systems (phantoms). It also lists data files or XML files that are found in the file systems that are not present in the Inventory database (orphans).
Report XML Discrepancies	Once per invocation of the utility		<i>EcDsAmCheckXMLArchive</i>	The XML Archive Check utility writes a discrepancy report to a text file. The report lists granules without XML files in the XML Archive. It also reports the XML files that are found in the XML Archive not present in the Inventory database.

#### 4.1.1.4 AIM interfaces with BMGT

The BMGT subsystem bypasses the AIM software modules and directly accesses the AIM Inventory database, XML Archive, and Granule Data File Archive. Access to the Inventory database is done through stored procedures in the Inventory database. Figure 4.1-7 illustrates these interfaces.



**Figure 4.1-7. AIM Interfaces with BMGT**

Table 4.1-8 describes each of the interfaces show above.

**Table 4.1-8. AIM Interfaces with BMGT (1 of 2)**

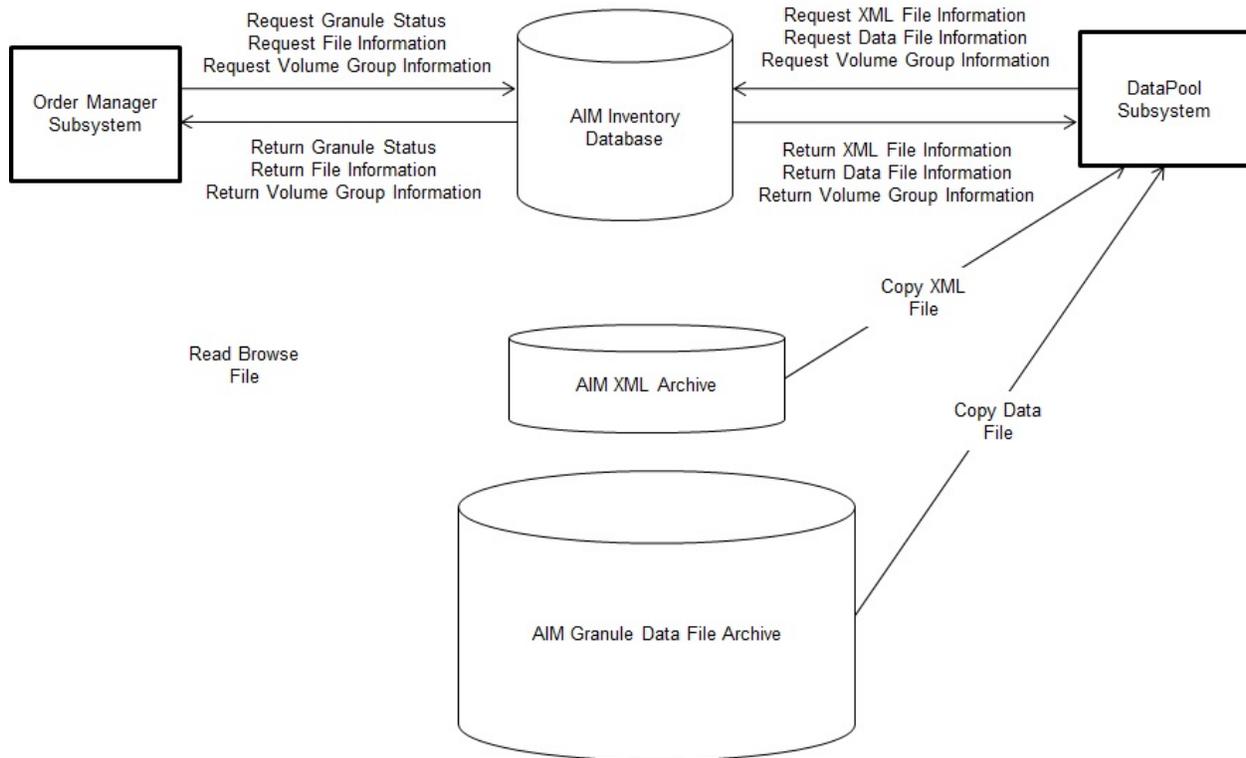
Event	Event Frequency	Interface	Initiated By	Event Description
Store BMGT Processing Data	Constant throughout running of BMGT	<i>Stored Procedures</i>	<i>BMGT processes</i>	The BMGT processes store persistent data related to their processing in the Inventory database. Please refer to the BMGT subsystem for a more complete explanation of the information stored.

**Table 4.1-8. AIM Interfaces with BMGT (2 of 2)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Provide Event Notifications		<i>Stored Procedures</i>	<i>ECS processes</i>	The Inventory database provides a table for ECS components to log events. The events are retained for a configured time period and cleaned up by an AIM script. These events are used by BMGT to identify the events that occurred within a given cycle (the events are copied to a BMGT table). It is important to configure the event retention time to be long enough for BMGT to capture the events before they are removed.
Return BMGT processing data		<i>Stored Procedures</i>	<i>BMGT processes</i>	The BMGT retrieves its processing information directly from the Inventory database using a series of stored procedures.
Provide Granule Linkage		<i>Stored Procedures</i>	<i>BMGT processes</i>	The BMGT obtains Science to Browse granule linkage information directly from the Inventory database.
Provide XML file location	Once per granule processed	<i>Stored Procedure</i>	<i>BMGT processes</i>	The BMGT obtains the location of the XML file within the XML Archive directly from the Inventory database.
Provide Browse file location		<i>Stored Procedure</i>	<i>BMGT processes</i>	The BMGT obtains the location of Browse files directly from the Inventory database.
Provide XML Files	Once per granule processed		<i>BMGT processes</i>	The BMGT reads XML files directly from the XML archive to produce the METG/C products.
Provide Descriptor files	Once per ESDT processed		<i>BMGT processes</i>	The BMGT reads Descriptor files directly from the archive to produce the METC products. These files are in ODL format, and BMGT uses the xmlsvcs framework to convert them to XML.

#### 4.1.1.5 AIM interfaces with the Order Manager and DataPool subsystems

Figure 4.1-8 shows the AIM context diagram for OMS and DPL. Table 4.1-9 shows the AIM interfaces with OMS and DPL.



**Figure 4.1-8. AIM Context Diagram (OMS and DPL)**

**Table 4.1-9. AIM Interfaces with OMS and DPL (1 of 3)**

Event	Event Frequency	Interface	Initiated By	Event Description
Request Granule Status	Once for each granule in an Order	Stored Procedure	EcOmOrderManager	The Order Manager server executes stored procedures to request the status information (deleted, active) for each granule ordered.
Request File Information	Once for each granule in an Order	Stored Procedure	EcOmOrderManager	The Order Manager server executes stored procedures to request file information, such as file size, for each granule ordered.

**Table 4.1-9. AIM Interfaces with OMS and DPL (2 of 3)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Request Volume Group information	Once per granule processed	<i>Stored Procedure</i>	<i>OMS</i>  <i>DPL DPAD</i>	The OMS executes stored procedures to request Volume Group information within the Inventory database.  The DataPool Action Driver requests information about the location of the Granule Data file from the Inventory database when staging a granule into the DPL.
Return Granule Status	Once for each granule in an Order	<i>Stored Procedure</i>	<i>EcOmOrderManager</i>	The Inventory database returns information about the status of the granule within the archive. The Inventory database also maps Local Granule IDs to internal granule IDs for OMS.
Return File Information		<i>Stored Procedure</i>	<i>EcOmOrderManager</i>	The Inventory database returns information such as file size, checksum, etc. about each granule ordered in OMS.
Return Volume Group Information		<i>Stored Procedure</i>	<i>EcOmOrderManager</i>	The Inventory database returns information about the location of Browse files within the AIM Granule Archive.
Read Browse File	Once per browse granule ordered		<i>EcOmOrderManager</i>	The OMS distributes browse file directly out of the Granule Archive.
Request XML File Information	Once per granule processed	<i>Stored Procedure</i>	<i>DPL DPAD</i>	The DataPool Action Driver requests information about the XML file from the Inventory database when staging a granule into the DPL.
Request Data file information	Once per granule processed	<i>Stored Procedure</i>	<i>DPL DPAD</i>	The DataPool Action Driver requests information about the Granule Data file from the Inventory database when staging a granule (Science or Browse) into the DPL.
Return XML File information	Once per granule processed	<i>Stored Procedure</i>	<i>DPL DPAD</i>	The Inventory database provides information such as file name and path to the DPAD.

**Table 4.1-9. AIM Interfaces with OMS and DPL (3 of 3)**

Event	Event Frequency	Interface	Initiated By	Event Description
Return Data File information	Once per granule processed	<i>Stored Procedure</i>	<i>DPL DPAD</i>	The Inventory database provides information such as file name (internal and distributed), checksum, file size, and LocalGranuleId to the DPAD.
Return Volume Group information	Once per granule processed	<i>Stored Procedure</i>	<i>DPL DPAD</i>	The Inventory database provides volume group information to the DPAD.
Copy XML File	Once per granule processed		<i>DPL DPAD</i>	The DPAD stages granules into the DataPool by copying the XML file directly from the XML Archive to the DataPool file system.
Copy Data File	Once per granule processed		<i>DPL DPAD</i>	The DPAD stages granules into the DataPool by copying the granule data file (Science or Browse) directly from the Granule Data File Archive to the DataPool file system.

#### 4.1.2 DSS Error Handling and Processing

The XML Validation Utility is a Java process and use a “pipe” based interface with the DPLIngest. They follow the coding standards for Java and contain only one log file. Requests and parameters are passed from the DPLIngest to the NDPIU and XVU by writing to the pipe. Spaces are used as delimiters between the arguments passed to these utilities. The final processing status of each request is written to the application log along with information to identify the granule being processed.

Both of these utilities return results to the DPLIngest by writing to the pipe. The NDPIU returns a value of 0 for success if all metadata was successfully inserted. In the case of an error, it returns a value of 1 along with a detailed text message describing the error. This utility doesn’t contain any persistent storage of previous requests; it assumes that DPLIngest will always link a specific granule ID to the same granule. Thus if it encounters a situation where metadata already exists for a given granule ID, the NDPIU assumes that it previously processed the granule and returns a success result.

The XVU has multiple possible return values. It returns a value of 0 for success; it returns a value of 2 if metadata in the request does not pass the validation process, it returns a value of 3 to indicate that the metadata passed validation but some optional elements were removed (this is considered a “warning” message), and it returns a value of 4 to indicate that errors were encountered attempting to validate the granule and that DPLIngest should try the request again. When the XVU returns 2, 3, or 4 it will also return a detailed text message describing the warning or error. The XVU has no persistent storage of requests, in the event that DPLIngest

validates the same granule more than one time, the XVU will process the XML metadata file without regard to previous any actions.

The ESDT Maintenance GUI responds directly to the operator, thus it displays error messages within the GUI and has a separate screen for displaying validation errors. It also has an application log for capturing processing information and error messages. The Granule Deletion utilities, the Quality Assurance Update utility, the XML Replacement utility, and the Archive Check utilities are command line utilities that interact directly with the Operator. They write to the operator console/xterm and use application logs to show detailed processing flow information and detailed error messages.

### 4.1.3 DSS Data Stores

Table 4.1-10 provides a description of the data stores for the AIM CSCI, and the conceptual model of the data store. The physical model for the AIM data stores can be found in the AIM Database Design and Schema Specifications for the EED Contract (CDRL 311).

**Table 4.1-10. AIM CSCI Data Stores (1 of 2)**

Data Store	Type	Description
AIM Inventory	Database	<p>The primary purpose of the AIM Inventory database is to configure ESDTs and track the status and location of granules within the DSS. The Inventory database catalogs information about the following objects:</p> <ul style="list-style-type: none"> <li>• ESDT definitions</li> <li>• Collection level information</li> <li>• Browse data</li> <li>• Science data (as granules)</li> <li>• Quality Assessments</li> <li>• Delivered Algorithm Packages</li> <li>• Production History</li> </ul>

**Table 4.1-10. AIM CSCI Data Stores (2 of 2)**

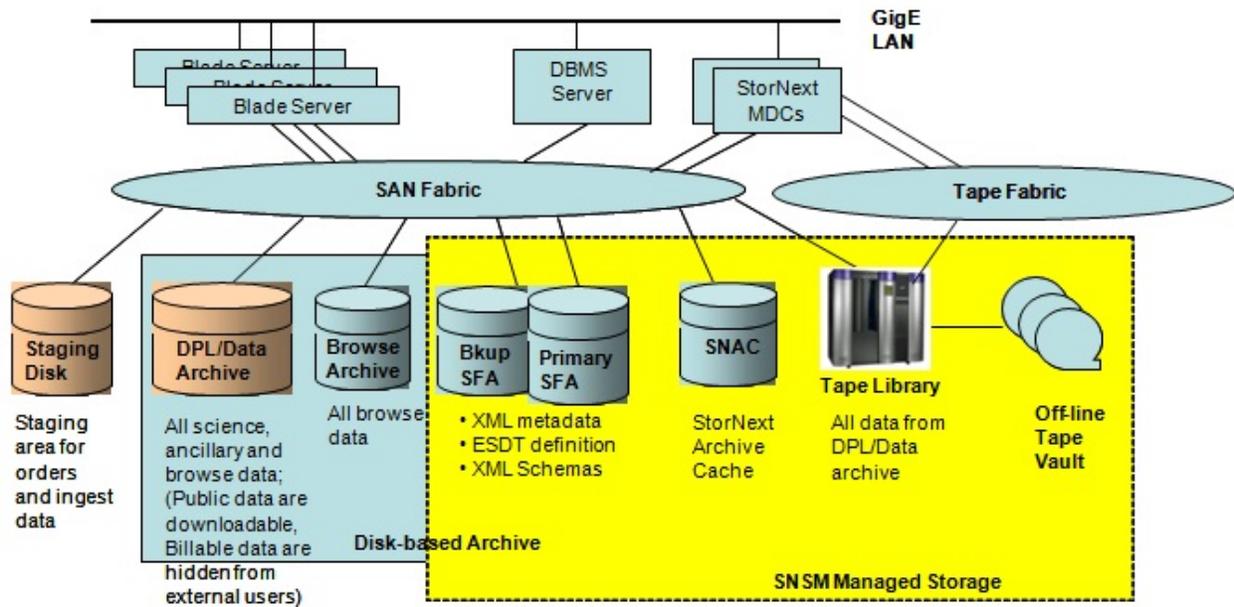
Data Store	Type	Description
XML Archive	File System	<p>The XML archive is a SAN attached managed file system that is available to most of the processing blades. It contains directories for the following:</p> <ul style="list-style-type: none"> <li>▪ A metadata directory stores an XML metadata file for each Science granule archived in the ECS. The directories for storing XML metadata are first separated by ESDT. Within each ESDT granule directories are partitioned into directories based upon the year and month of the acquisition time. If acquisition time is not recorded for an ESDT, then the insert time of the granule will be used.</li> <li>▪ A descriptor directory stores a descriptor file along with the associated XML schema file for the ESDT installed in the mode.</li> <li>▪ An MCF directory stores a Metadata Control File (MCF) for each ESDT installed in the system.</li> <li>▪ A BMGT directory is used by the BMGT subsystem for persistent storage of production outputs and reports.</li> </ul>
Granule Archive	File System	<p>The Granule Archive is the primary data store for the ECS. It consists of a set of managed file systems that are broken up into directories that are assigned to the storage of files for a given ESDT or set of ESDTs. These directories are referred to as Volume Groups and typically have a specific time range of granules that are stored in the directory.</p>

## 4.2 DPL Ingest Subsystem Overview

The Data Pool Ingest service handles the SIPS ingest interface<sup>1</sup>, cross-DAAC ingest, EDOS ingest, ASTER Ingest and Polling without Delivery Record specifically for EMOS. Unlike the classic INGST subsystem, the Data Pool Ingest service will insert the ingested data into the Data Pool, in addition to inserting the ingested data into the archive.

### DPL Ingest Subsystem Context

Figure 4.2-1 is the DPL Ingest context diagram. The diagram provides an illustration of the Data Pool Ingest and archiving steps. Table 4.2-1 provides descriptions of the interface events shown in the DPL Ingest Subsystem context diagram.



**Figure 4.2-1. DPL Ingest Subsystem Context Diagram**

<sup>1</sup> EOSDIS Document 423-41-57 Interface Control Document between the ECS and the SIPS Volume 0 revision H

**Table 4.2-1. DPL Ingest Subsystem Interface Events**

Event	Interface Event Description
Create PDR	SIPS providers place their data and PDR files into a polling directory. The directory can be local, e.g., accessible via a mount point; or remote, i.e., accessible via FTP/SCP. For Polling without Delivery Record provider, the data file is transferred to the polling directory rather than a PDR.
Polling	The DPL Ingest Service polls these directories as configured by the DAAC.
Request Queue	The DPL Ingest Service queues ingest requests for validation and processing. The DPL Ingest Service queues all PDR that it finds. To decide which validated PDR will be processed next, it checks available resources, timestamps and priorities of the requests.
ODL To XML Conversion	The DPL Ingest Service converts the input ODL metadata (if applicable) to XML metadata for insertion into DPL.
XML Metadata Validation	The DPL Ingest Service calls the XML Validation Utility to validate the XML metadata file.
DPL Insert	The granule files are copied into the Data Pool SAN, using hidden directories for that purpose unless the DAAC requested that the data be published on insert.
Archive	The DPL Ingest Service then copies the granules from the hidden Data Pool directories into the StorNext archive.
Update Archive Information	The DPL Ingest Service updates archive information in the AIM database.
SSS notification	The DPL Ingest service places a record for the Spatial subscription server to decide whether any subscription should fire based on the granule insert.
Notify with PAN or PDRD	The provider is notified of the ingest outcome in the format of PDRD (if PDR validation failed) or PAN (for reasons other than PDR validation failure).
Queue for Publication	The DPL Ingest service places a record for the granule to be published if its collection is marked for publication. The DPAD will then perform the publication.

### **DPL Ingest Subsystem Structure**

The DPL Ingest Subsystem consists of four CSCIs: the Polling Service, the Processing Service, the Notification Service, and the DPL Ingest GUI. The Polling Service is responsible for the provision of work to the service via transferring Product Delivery Records (PDRs) into the system and registering them, or in the case of Polling without Delivery record, creating a PDR for each data file and registering them. The Processing Service picks up registered PDRs and attempt to ingest the data they describe into the Data Pool and the Archive, performing any additional processing required for specific inventory. The Processing component will checkpoint a particular PDR on completion of various steps during processing and register a notification (i.e. PDRD or PAN) for Notification Service to process when the PDR reached a terminal state. Terminal states are Successful, Partial\_Failure, Failed, Cancelled, and Partially\_Cancelled. Terminal states are conveyed to the provider by means of a Product Acceptance Notification (PAN) or Product Delivery Discrepancy Report (PDRD). The Notification Service will detect registered notifications and deliver them to the provider based on the provider configured notification methods. The DPL Ingest GUI is used to configure, monitor and control the operations of the DPL Ingest Service.

## Use of COTS in the DPL Ingest Subsystem

- RogueWave's Tools.h++

The Tools.h++ class libraries are used by the DPL Ingest Service to provide basic functions and objects such as strings and collections. These libraries must be installed with the DPL Ingest software for any of the DPL Ingest Service processes to run.

- PostgreSQL libpq

The PostgreSQL libpq provides access between DPL Ingest Service custom code and the PostgreSQL backend server.

- PostgreSQL Server

The PostgreSQL server provides access for DPL Ingest Service to insert, update and delete DPL Ingest Requests, DPL Ingest configurations, and Operator Interventions. The PostgreSQL Server must be running during operations for the DPL Ingest Service to process DPL Ingest Requests.

- UNIX Network Services

DNS, NFS, E-mail, FTP, TCP/IP and the other UNIX services provided are obtained from the CSS.

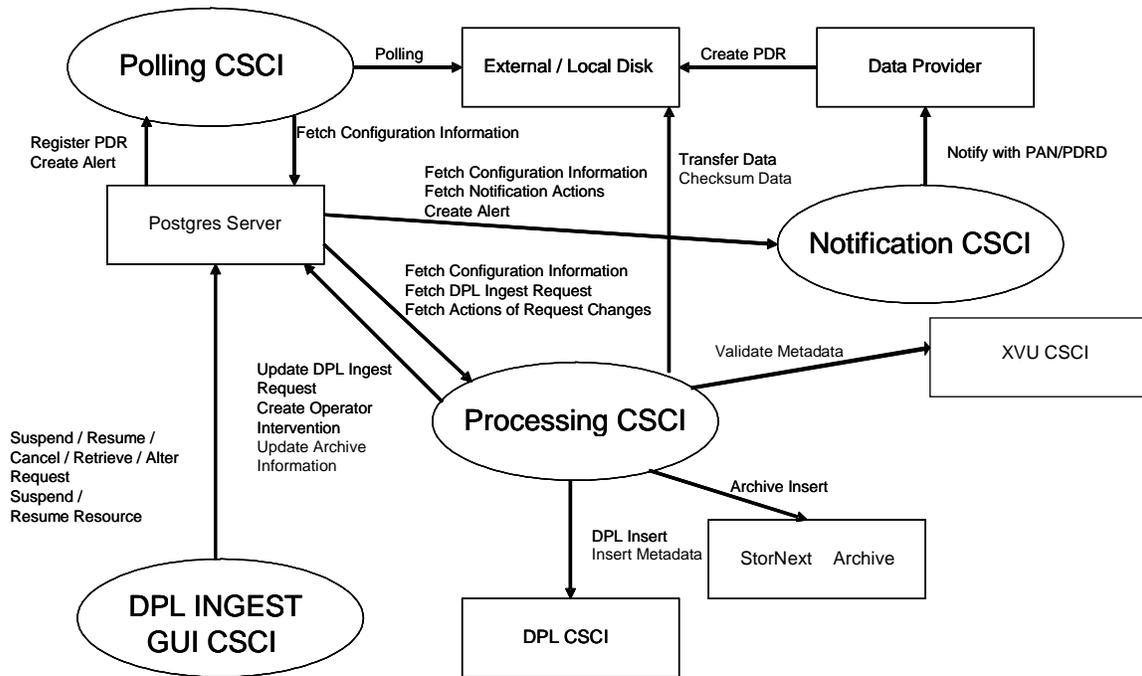
### 4.2.1 DPL Ingest Computer Software Configuration Item Description

#### 4.2.1.1 DPL Ingest Service CSCI Functional Overview

The DPL Ingest Subsystem consists of four CSCIs: the Polling Service, the Processing Service, the Notification Service, and the DPL Ingest GUI. The Processing Service executes as a process and interacts with the following CSCIs: INGEST Database, the Polling Service, the Notification Service, the XML Validation Utility (XVU), and the Data Pool System (DPL). The Polling Service transfers Product Delivery Records (PDRs) into the system and registers them to the INGEST Database. Processing Service retrieves the PDRs from INGEST Database and validates them. If the PDR is valid, Processing Service attempts to ingest the inventory they describe into the Data Pool and Archive, performing any additional processing required for specific inventory. The Processing Service updates the status of a particular PDR on completion of various steps during processing. For an invalid PDR, a PDRD is generated. An operator intervention is created if the request encounters a processing problem. DAAC OPS personnel can use the DPL Ingest GUI to monitor and control the processing of the request. In response to an intervention, the Operator can view the error details, retry the erroneous granule or fail the request if the problem cannot be resolve through retries. Processing Service also generates notification for each request that has reached a terminal state and register notification in INGEST Database. The Notification Service will detect the registered notifications and deliver them to the provider based on the provider configured notification methods. Operator Alert is generated when an internal or external resource failure is detected. When an operator alert is generated, DPL Ingest services will halt dispatching of the requests that are utilizing those failed resources, retries the failed operation that caused the alert (if so configured) and automatically clears the alert if the operation succeeds on retry. Operator can view the operator alerts on DPL Ingest GUI and can manually clear the operator alerts through the GUI.

### 4.2.1.2 DPL Ingest Service CSCI Context

Figure 4.2-2 is the DPL Ingest Service CSCI context diagrams. The diagrams show the events sent to the DPL Ingest Service CSCIs and the events the DPL Ingest CSCIs send to other CSCIs. Table 4.2-2 provides descriptions of the interface events shown in the DPL Ingest Service CSCI context diagram.



**Figure 4.2-2. DPL Ingest CSCI Context Diagram**

**Table 4.2-2. DPL Ingest CSCI Interface Events (1 of 2)**

Event	Interface Event Description
Create PDR	<b>SIPS providers</b> place their data and PDR files into a polling directory which will be polled by the Polling CSCI. The directory can be local, e.g., accessible via a mount point; or remote, i.e., accessible via FTP.
Polling	The Polling CSCI polls PDRs from directories ( <b>External/Local Disk</b> ) by <b>Data Provider</b> as configured by the DAAC.
Register PDR	The Polling CSCI queues ingest requests for validation and processing into the <b>PostgreSQL Server (INGST Database)</b> . The Processing CSCI later queues all PDR that it finds. To decide which validated PDR will be processed next, it checks available resources and DAAC configured priorities.
ODL To XML Conversion	The DPL Ingest Service converts the input ODL metadata (if applicable) to XML metadata for insertion into DPL.
XML Metadata Validation	The DPL Ingest Service calls the <b>XML Validation Utility</b> to validate the XML metadata file.
DPL Insert	The Processing CSCI copies the granule files into the <b>Data Pool SAN</b> , using hidden directories for that purpose unless the DAAC requested that the data be published on insert.
Archive Insert	The Processing CSCI copies the granule files in to the <b>StorNext archive</b> .
Update Archive Information	The DPL Ingest Service updates archive information in the AIM database.
SSS notification	The DPL Ingest service places a record in the <b>SSS database</b> for the Spatial subscription server to decide whether any subscription should fire based on the granule insert.
Notify with PAN or PDRD	The Notification CSCI sends notification to the <b>Data Provider</b> , it could be immediate via PDRD if PDR validation failed, or later on via a short or long PAN.
Queue for Publication	The DPL Ingest service places a record in the <b>AIM database</b> for the granule to be published if its collection is marked for publication. The DPAD will then perform the publication.
Create Alert	The Polling CSCI, Processing CSCI and Notification CSCI creates an alert for resource failures and stores the alert in the <b>PostgreSQL Server (INGST Database)</b> .
Fetch Config Info	The Polling CSCI, Processing CSCI and Notification CSCI retrieves the configuration information from <b>PostgreSQL Server (INGST Database)</b> .
Update DPL Ingest Request	The Processing CSCI updates DPL Ingest request in the <b>PostgreSQL Server (INGST Database)</b> .
Create Operator Intervention	The Polling CSCI, Processing CSCI creates new Operator Intervention for request failures in the <b>PostgreSQL Server (INGST Database)</b> .
Fetch DPL Ingest Request	The Processing CSCI retrieves information associated with a DPL Ingest request from the <b>PostgreSQL Server (INGST Database)</b> .
Fetch Actions of Request Changes	The Processing CSCI retrieves actions regarding request changes, such as, request priority change, cancel request, suspend request, and update request parameters from the <b>PostgreSQL Server (INGST Database)</b> .
Validate Metadata	The Processing CSCI populates the metadata files and sends them to the <b>XVU CSCI</b> for validation.

**Table 4.2-2. DPL Ingest CSCI Interface Events (2 of 2)**

Event	Interface Event Description
Insert Metadata	The Processing CSCI sends the granule metadata to the <b>DPL CSCI</b> for insertion into the AIM database.
Fetch Notification Actions	The Notification CSCI retrieves actions regarding request notifications from the <b>PostgreSQL Server (INGST Database)</b> .
Suspend/Resume/Cancel/Alter/Retrieve Request	The DPL Ingest GUI CSCI suspends, resumes, cancels, alters and retrieves requests from the <b>PostgreSQL Server (INGST Database)</b> .
Suspend/Resume Resource	The DPL Ingest GUI CSCI suspends or resumes dispatching to all or selected resources in the <b>PostgreSQL Server (INGST Database)</b> .
Transfer Data	The Processing CSCI transfers data files from the <b>External/Local Disk</b> specified in PDR.
Checksum Data	The Processing CSCI checksum data files from the <b>checksum information</b> specified in PDR or calculate the checksum based on the provider and system configuration.

#### 4.2.1.3 DPL Ingest Architecture

The Polling Ingest Interface (EcDIInPollingService) polls accessible file system locations to detect data to be ingested. This process submits a Product Delivery Record (PDR). The Cross-DAAC Ingest Interface (EcDIInEmailGWServer) polls a configured directory for distribution notices (flat files converted from email messages). This process detects the distribution notice files and creates Product Delivery Record files, which are put in a polling directory and polled by the Polling Ingest Interface.

The Polling Ingest Interface queues ingest requests into the PostgreSQL Server (INGST database) for Processing Service to pick up. The Processing Interface queues all PDR that it finds, to decide which validated PDR will be processed next, it checks available resources, timestamps and priorities of the request. The Processing Interface validates metadata using the XVU and inserts the granules into the AIM inventory. The Processing Interface copies the granule files into Data Pool SAN, using hidden directories for that purpose unless the DAAC requested that the data be published on insert. The processing Interface copies the granule files into the StorNext archive. The processing service copies the XML metadata file to the small file archive.

Figure 4.2-3 is the DPL Ingest CSCI architecture diagram. The diagram shows the events sent to the DPL Ingest CSCI processes and the events the DPL Ingest CSCI processes send to other processes.

**Note: System startup and shutdown** - Please refer to the release-related, current version of the Mission Operations Procedures for the EED Contract document (611) and the current EED Contract Training Material document (625).

#### 4.2.1.4 DPL Ingest Process Descriptions

Table 4.2-3 provides the descriptions of the processes shown in the DPL Ingest CSCI architecture diagram (Figure 4.2-3).

**Table 4.2-3. DPL Ingest CSCI Processes (1 of 2)**

Process	Type	Hardware CI	Source	Functionality
EcDInPollingService	Server	DPLHW	Developed	<ul style="list-style-type: none"> <li>• Detect new Product Delivery Records (PDRs) and transfer them into system.</li> <li>• Creates a unique identifier for the request.</li> <li>• Register request.</li> </ul>
EcDInGui	GUI	INTHW	Developed	Provides Maintenance and Operations (M&O) personnel the capability, via GUI Interface, <ul style="list-style-type: none"> <li>• To modify ingest configuration parameters.</li> <li>• To monitor the status of ongoing ingest requests, to suspend, resume, cancel, alter or retrieve DPL ingest requests.</li> <li>• To suspend or resume resource.</li> </ul>
EcDInProcessingService	Server	DPLHW	Developed	<ul style="list-style-type: none"> <li>• Ingests granules associated with ingest requests (PDRS) into the Datapool and archive.</li> <li>• Registers granule information with AIM</li> <li>• Manages the DPL ingest request traffic and the processing of the DPL ingest requests.</li> <li>• Provides the capability to process multiple ingest requests concurrently by placing the request in a queue.</li> <li>• In the event of a failure, the EcDInProcessingService process restores on-going requests from the Ingest database.</li> </ul>
EcDInNotificationService	Server	DPLHW	Developed	<ul style="list-style-type: none"> <li>• Send the end-user Notification, either Product Acceptance (PAN) or Product Delivery Discrepancy Report (PDRD), on completing a ingest request.</li> </ul>

**Table 4.2-3. DPL Ingest CSCI Processes (2 of 2)**

Process	Type	Hardware CI	Source	Functionality
PostgreSQL	Server	ACMHW	COTS	<ul style="list-style-type: none"> <li>Stores and provides access to the DPL Ingest Service internal data. In particular, the database stores the Ingest operations databases – DPL Ingest History Logs and the DPL Ingest request checkpoint state, and template information. See Section 4.2.1.6 DPL Ingest Data Stores.</li> </ul>

EMD Baseline Information System (EBIS) Document 920-TDx-001 (Hardware Design Diagram) provides descriptions of the HWCI, and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

#### 4.2.1.5 DPL Ingest Process Interface Descriptions

Table 4.2-4 provides descriptions of the interface events shown in the DPL Ingest CSCI Architecture diagram.

**Table 4.2-4. DPL Ingest CSCI Process Interface Events (1 of 5)**

Event	Event Frequency	Interface	Initiated By	Event Description
Create PDR	One per request	Directories on remote or local disk	External Data Provider	SIPS providers place their data and PDR files into a polling directory which will be polled by the EcDlInPollingService. The directory can be local, e.g., accessible via a mount point; or remote, i.e., accessible via FTP/SCP.
Polling	One per request	Directories on remote or local disk	<i>Process:</i> EcDlInPollingService <i>Class:</i> DpInPoller	The EcDlInPollingService polls PDRs from directories (External/Local Disk) by Data Provider as configured by the DAAC.

**Table 4.2-4. DPL Ingest CSCI Process Interface Events (2 of 5)**

Event	Event Frequency	Interface	Initiated By	Event Description
Register PDR	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDInPollingService <i>Class:</i> DpInPoller	The EcDInPollingService queues ingest requests for validation and processing into the PostgreSQL Server (INGST database). The EcDInProcessingService later queues all PDR that it finds. To decide which validated PDR will be processed next, it checks available resources and DAAC configured priorities.
DPL Insert	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDInProcessingService <i>Class:</i> DpInDplRegistrationQAction	The EcDInProcessingService copies the granule files into the Data Pool SAN, using hidden directories for that purpose unless the DAAC requested that the data be published on insert.
Archive Insert	One per request	<i>Process:</i> StoreNext copy	<i>Process:</i> EcDInProcessingService <i>Class:</i> DpInArchiveQAction	The EcDInProcessingService copies the granule files into the StoreNext archive file system.

**Table 4.2-4. DPL Ingest CSCI Process Interface Events (3 of 5)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Update Archive Information	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDlInProcessingService <i>Class:</i> DpInInventoryInsertQAction	The EcDlInProcessingService updates archive information in the AIM database.
Notify with PAN or PDRD	One per email notification request	<i>Process:</i> Sendmail (COTS) Ftp daemon (COTS)	<i>Process:</i> EcDlInNotificationService <i>Class:</i> DpInNotifyEmailAction DpInNotifyFtpAction	The EcDlInNotificationService sends notification to the Data Provider, it could be immediate via PDRD if PDR validation failed, or later on via a short or long PAN.
Create Alert	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDlInPollingService EcDlInProcessingService EcDlInNotificationService <i>Class:</i> DpCoAlert	The EcDlInPollingService, EcDlInProcessingService and EcDlInNotificationService create an alert for resource failures and stores the alert in the PostgreSQL Server (INGST Database).
Fetch Config Info	One per startup/ One per configurable interval	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDlInPollingService EcDlInProcessingService EcDlInNotificationService <i>Class:</i> DpInNotifyDatabase	The EcDlInPollingService, EcDlInProcessingService and EcDlInNotificationService retrieve the configuration information from PostgreSQL Server (INGST Database).
Update DPL Ingest Request	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDlInProcessingService <i>Class:</i> DpInProcessingDbInterface	The EcDlInProcessingService updates DPL Ingest request in the PostgreSQL Server (INGST Database).

**Table 4.2-4. DPL Ingest CSCI Process Interface Events (4 of 5)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Create Operator Intervention	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDInProcessingService <i>Class:</i> DpInProcessingDbInterface	The EcDInProcessingService creates new Operator Intervention for request failures in the PostgreSQL Server (INGST Database).
Fetch DPL Ingest Request	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDInProcessingService <i>Class:</i> DpInProcessingDbInterface	The EcDInProcessingService retrieves information associated with a DPL Ingest request from the PostgreSQL Server (INGST Database).
Fetch Actions of Request Changes	One per configurable interval	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDInProcessingService <i>Class:</i> DpInProcessingDbInterface	The EcDInProcessingService retrieves actions regarding request changes, such as, request priority change, cancel request, suspend request, and update request parameters from the PostgreSQL Server (INGST Database).
ODL To XML Conversion	One per ODL metadata	<i>Process:</i> OdlToXmlConverter <i>Library:</i> odlToXml.jar	<i>Process:</i> EcDInProcessingService <i>Class:</i> DpInGranuleScheduler	The EcDInProcessingService invoke a java utility to convert the ODL metadata file into XML metadata file.
Validate XML Metadata	One per metadata validation	<i>Process:</i> EcDsAmXvu <i>Library:</i> xmlsvcs	<i>Process:</i> EcDInProcessingService <i>Class:</i> DpInXmlValidationQAction	The EcDInProcessingService populates the metadata files and sends them to the XVU for validation.

**Table 4.2-4. DPL Ingest CSCI Process Interface Events (5 of 5)**

Event	Event Frequency	Interface	Initiated By	Event Description
Request Data Search	One per granule pointer in linkage file	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDlInProcessingService <i>Class:</i> DpInGranuleScheduler	The EcDlInProcessingService queries PostgreSQL server (Inventory database), for the granule corresponding to a particular ESDT short name and version, which has a particular local granule id.
Fetch Notification Actions	One per configurable interval	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDlInNotificaionService <i>Class:</i> DpInNotifyDatabase	The EcDlInNotificationService retrieves actions regarding request notifications from the PostgreSQL Server (INGST Database).
Suspend/Resume/Cancel/Alter/Retrieve Request	One per click	<i>Process:</i> PostgreSQL Server (COTS)	DPL Ingest GUI	The DPL Ingest GUI scripts send suspend, resume, cancel, alter and retrieve request command to the PostgreSQL Server (INGST Database).
Suspend/Resume Resource	One per click	<i>Process:</i> PostgreSQL Server (COTS)	DPL Ingest GUI	The DPL Ingest GUI scripts send, suspend, or resume resource command to the PostgreSQL Server (INGST Database).
Transfer Data	One per science data file activity	<i>Process:</i> Ftpd (COTS) or sshd (COTS)	<i>Process:</i> EcDlInProcessingService <i>Class:</i> DpInInternalFtpTransferQAction/DpInInternalScpTransferQAction	The EcDlInProcessingService transfers data files from the External/Local Disk specified in PDR.
Checksum Data	One per data file activity	<i>Process:</i> Checksum utilities (COTS)	<i>Process:</i> EcDlInProcessingService <i>Class:</i> DpInChecksumQAction	The EcDlInProcessingService checksums data files from the checksum information specified in PDR or calculate the checksum based on the provider and system configuration.

#### 4.2.1.6 Ingest Data Stores

The DPL Ingest CSCI uses the COTS product PostgreSQL to store related DPL Ingest Information. Table 4.2-5 provides descriptions of the data stores.

**Table 4.2-5. DPL Ingest CSCI Data Stores**

<b>Data Store</b>	<b>Type</b>	<b>Description</b>
INGST Database	PostgreSQL	INGST Database is designed to store the persistent information of user request, processing configuration, request aging configuration, and request cleanup configuration.
Inventory Database	PostgreSQL	The AIM Inventory database is designed to store the minimal metadata information of ingested granules and tape archive file storage information of data and metadata files. It also implements the large majority of the persistent data requirements for the DPL subsystem which supports large online cache of important ECS data at each DAAC and avoids tape access to ECS archive.

### **4.3 Client Subsystem Overview - DELETED**

### **4.4 Data Management Subsystem Overview**

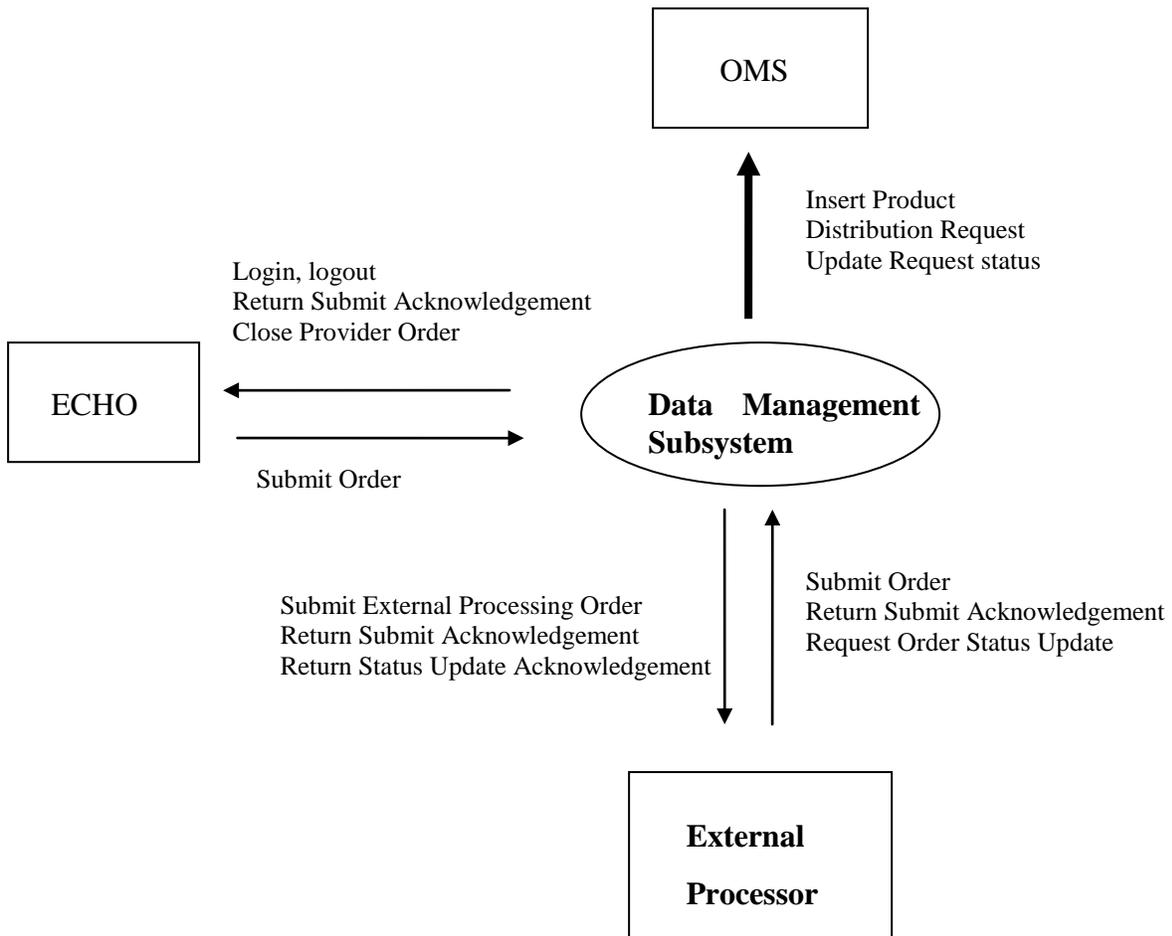
The Data Management Subsystem (DMS) provides interoperability between the ECHO (EOS ClearingHouse) and the ECS. The DMS provides this service by supplying a gateway process. The ECHO WSDL (Web Service Definition Language) Order Component (EWOC) allows users to order ECS products via order tools that interface with ECHO. The ECS will no longer support search and browse capabilities as these will be handled internally by ECHO. The EWOC will provide the means for ECHO to present orders to ECS on behalf of the user.

DMS functionality includes:

- DMS validates and places orders that users submit using the clients that interface with ECHO.
- DMS submits orders to External Processors for granules that require external processing.
- DMS allows External Processors to update the status of requests in ECS.

#### **Data Management Subsystem Context**

Figure 4.4-1 is the Data Management Subsystem context diagrams. The diagrams show the events sent to the Data Management Subsystem and the events the Data Management Subsystem sends to other external systems and CSMS subsystems. Table 4.4-1 provides descriptions of the interface events shown in the Data Management Subsystem context diagrams.



**Figure 4.4-1. Data Management Subsystem Context Diagram**

**Table 4.4-1. Data Management Subsystem Interface Events**

<b>Event</b>	<b>Interface Event Description</b>
Insert Product Distribution Request	The Data Management Subsystem (DMS) inserts product distribution requests in the Order Manager Data Base Management System within the <b>Order Manager Subsystem (OMS)</b> .
Update Request Status	The DMS submits a request status update to the <b>OMS</b>
Submit Order	The DMS receives product requests from the <b>External Processor*</b> .
Return Submit Acknowledgement	The DMS receives confirmation of an external processing order from the <b>External Processor</b> .
Request Status Update	The DMS receives status update requests from the <b>External Processor</b> .
Submit External Processing Order	The DMS submits an external processing request in the <b>External Processor</b> .
Return Submit Acknowledgement	The DMS returns a confirmation of an order submitted from <b>External Processor</b> .
Return Status Update Acknowledgement	The DMS returns a confirmation of a status update request by the <b>External Processor</b> .
Submit Order	The DMS receives product requests from the <b>ECHO</b> on behalf of an external ECS user.
Login, logout	The DMS logs in and logouts to Authentication Service at <b>ECHO</b> to obtain security token.
Return Submit Acknowledgement	The DMS returns a confirmation of order submitted from <b>ECHO</b> .
Close Provider Order	The DMS updates the status of requests at <b>ECHO</b> when the requests reach terminal states at ECS.

*\*Note: For the purpose of this document, "External Processor" refers to either an External Subsetter (HSA) or an On-Demand Processor (S4PM), both of which are treated the same by the DMS.*

### **Data Management Subsystem Structure**

The DMS consist of one CSCI:

- The ECHO WSDL Order Component (EWOC) is a software configuration item. The EWOC provides a gateway between ECHO and ECS by allowing users using external client to submit ECS orders through ECHO. The EWOC validates and submits orders to ECS for product distribution. The EWOC also allows interaction with External Processors. External Processors receive external processing orders from the EWOC and submit status update requests to the EWOC.

### **Use of COTS in the Data Management Subsystem**

- Apache Axis 1.4

The Apache Axis packages are used to generate JAVA web service which uses SOAP messages for communication.

## **Error Handling and processing**

EMD Process Framework package is used for general error reporting. The functions can catch exceptions coming from try blocks and the exception stack trace is logged in the log files. Exceptions that occur during interaction with ECHO will be propagated to the ECHO to indicate order status to the user.

There are three kinds of logs: operations, debugging and performance.

Each conforms to and is supported by the process framework package under `/ecs/formal/COMMON/java/gov/nasa/emd/processframework` which wraps the Java Logging utility. Each type of log provides for four different levels of output: NONE, INFORMATION, VERBOSE and XVERBOSE.

For writing messages to the log, the following function from LogWrapper class is used:

```
Public static void log(int level, boolean debug, boolean ops, String message),
```

For example, the following writes to operations log with output level of INFORMATION.

```
LogWrapper.log (Logger.INFORMATION, false, true, "EWOC Initialization complete");
```

For writing messages to the debug log, the following function calls are used:

```
LogWrapper.log (Logger.VERBOSE, true, false, "CloseRequestPoller");
```

### **4.4.1 ECHO WSDL Order Component Software Description**

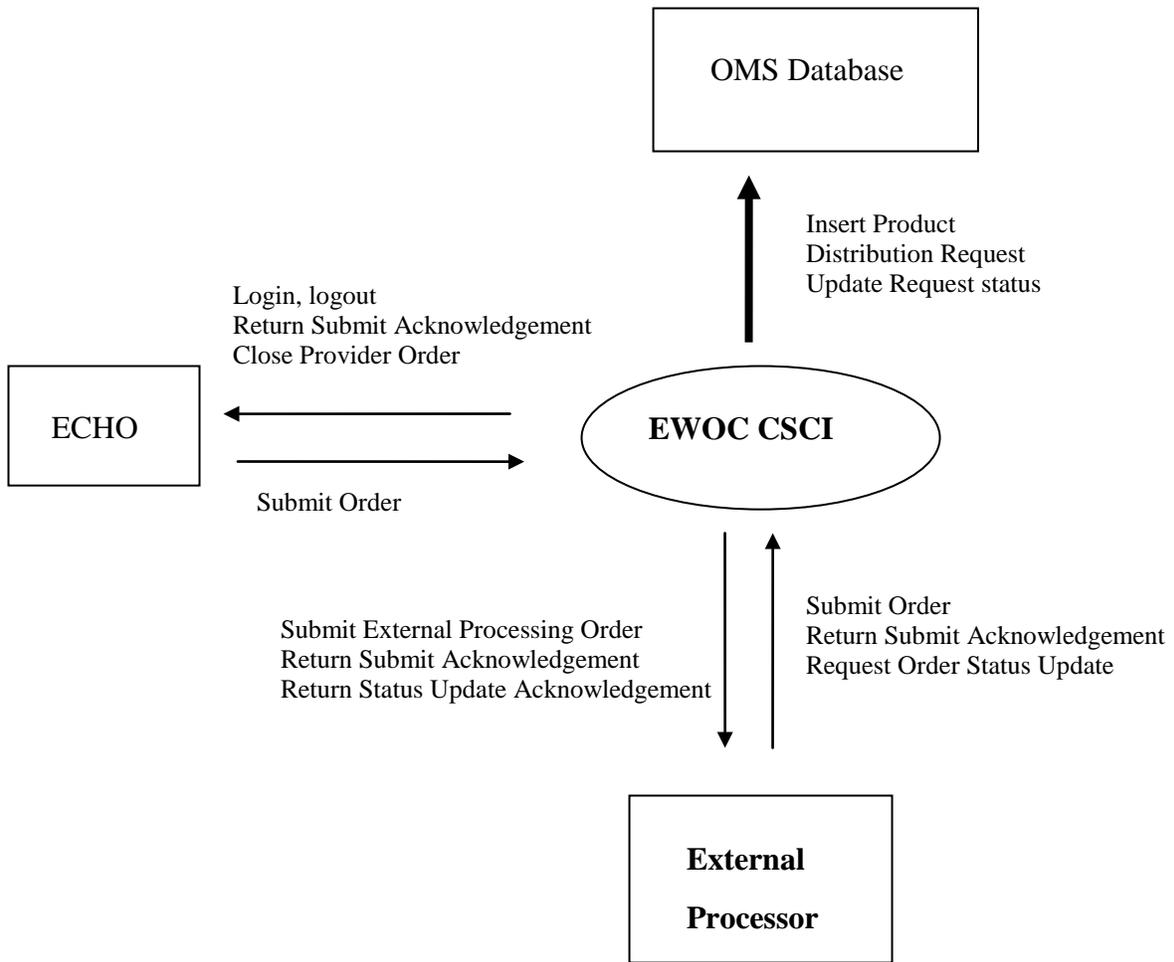
#### **4.4.1.1 ECHO WSDL Order Component Functional Overview**

The ECHO WSDL Order Component (EWOC) CSCI provides a gateway between ECHO and ECS systems. The users using the client will search, browse and order data using ECHO and submit orders to the EWOC CSCI. The EWOC CSCI then validates the order according to the ECS rules, and submits the requests to the Order Manager Subsystem for product distribution. The EWOC CSCI also updates the status of the request at ECHO to provide the user with an order status.

The EWOC CSCI is a web service. Apache Axis handles service layer and receives messages via SOAP format. Once the order is received, the EWOC returns submit acknowledgement which describes whether order submission was successful. For orders that require external processing, the EWOC places a request at the External Processor and accepts status update requests from the External Processor.

#### **4.4.1.2 ECHO WSDL Order Component Context**

Figure 4.4-2 is the EWOC CSCI context diagrams. The diagrams show the events sent to other CSCIs or CSCs and the events the EWOC CSCI receives from other CSCIs and CSCs. Table 4.4-2 provides descriptions of the interface events shown in the EWOC CSCI context diagrams.



**Figure 4.4-2. ECHO WSDL Order Component CSCI Context Diagram**

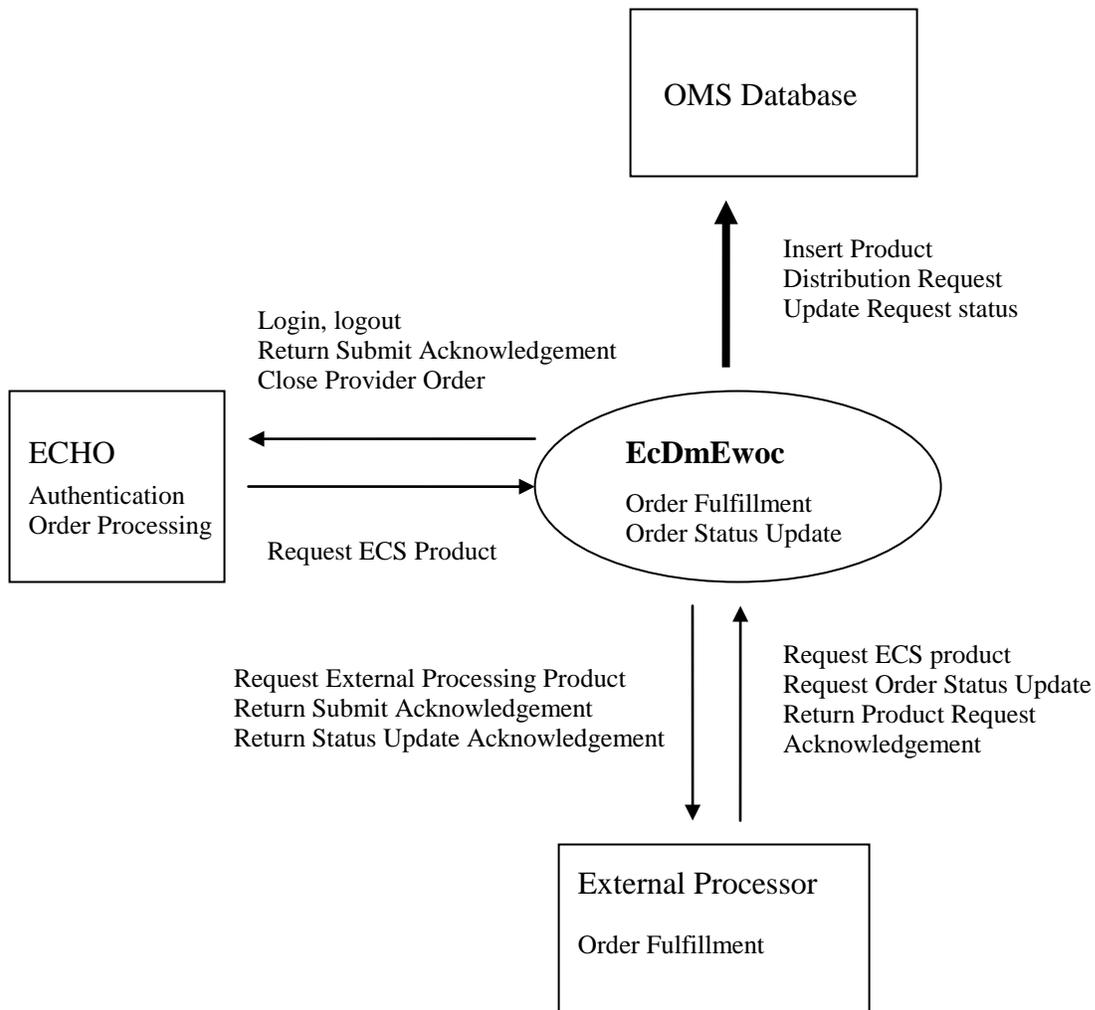
**Table 4.4-2. ECHO WSDL Order Component CSCI Interface Events**

Event	Interface Event Description
Insert Product Distribution Request	The ECHO WSDL Ordering Component (EWOC) inserts product distribution requests in the Order Manager Data Base Management System within the <b>Order Manager Subsystem (OMS)</b> .
Update Request Status	The EWOC submits a request status update to the <b>OMS</b> .
Submit Order	The EWOC receives product requests from the <b>External Processor</b> .
Return Submit Acknowledgement	The EWOC receives a confirmation of an external processing order from the <b>External Processor</b> .
Request Order Status Update	The EWOC receives status update requests from the <b>External Processor</b> .
Submit External Processing Order	The EWOC submits an external processing request in the <b>External Processor</b> .
Return Submit Acknowledgement	The EWOC returns a confirmation of an order submitted from <b>External Processor</b> .
Return Status Update Acknowledgement	The EWOC returns a confirmation of a status update request by the <b>External Processor</b> .
Submit Order	The EWOC receives product requests from the <b>ECHO</b> on behalf of an external ECS user
Login, logout	The EWOC logs in and logouts to Authentication Service at <b>ECHO</b> to obtain security token.
Return Submit Acknowledgement	The EWOC returns a confirmation of order submitted from <b>ECHO</b> .
Close Provider Order	The EWOC updates the status of requests at <b>ECHO</b> when the requests reach terminal states at ECS.

#### 4.4.1.3 ECHO WSDL Order Component Architecture

Figure 4.4-3 is the EWOC CSCI architecture diagram. The diagram shows the events sent to the EWOC CSCI processes and the events the EWOC CSCI process sends to other processes.

The EWOC CSCI is one process, the EcDmEwoc as shown in the ECHO WSDL Order Component CSCI Architecture Diagram.



**Figure 4.4-3. ECHO WSDL Order Component CSCI Architecture Diagram**

#### 4.4.1.4 ECHO WSDL Order Component Process Descriptions

Table 4.4-3 provides descriptions of the processes shown in the EWOC CSCI architecture diagram.

**Table 4.4-3. EWOC CSCI Processes**

Process	Type	Hardware CI	COTS/ Developed	Functionality
EcDmEwoc	Web Service	DMGHW	Developed	The ECHO WSDL Order Component is a web service that runs on Tomcat/Apache. The EWOC offers two basic interfaces. OrderFulfillment Port: External systems such as ECHO or External Subsetter can submit orders to this endpoint. The EWOC validates the order according to ECS rules and then bundles the order into separate requests before submitting the requests to OMS for product distribution. If the order is an external processing order, the EWOC submits an order to the external processor on behalf of ECS. OrderStatusUpdate Port: External processors can submit requests to update the status of an order using this interface.

**4.4.1.5 ECHO WSDL Order Component Process Interface Descriptions**

Table 4.4-4 provides descriptions of the interface events shown in the EWOC CSCI architecture diagram.

**Table 4.4-4. EWOC CSCI Process Interface Events (1 of 3)**

Event	Event Frequency	Interface	Initiated By	Event Description
Insert Product Distribution Request	One per product request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDmEwoc  <i>Classes:</i> OmsDataAccessImpl	The EcDmEwoc inserts product distribution requests into the Order Manager DBMS by invoking OMS stored procedures.
Update Request Status	One per status update request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcDmEwoc	The EcDmEwoc updates the request status in the MSS database by invoking OMS stored procedure.

**Table 4.4-4. EWOC CSCI Process Interface Events (2 of 3)**

Event	Event Frequency	Interface	Initiated By	Event Description
Request ECS Product	One per product request	<i>Process:</i> OrderFulfillment Port  <i>Classes:</i> OrderFulfillmentPortBindingImpl	<i>Process:</i> External Processor, ECHO	The External Processor or the ECHO submits an order through EWOC's OrderFulfillment Port.
Request Order Status Update	One per status update request	<i>Process:</i> OrderStatusUpdate Port  <i>Class:</i> OrderServiceImpl	<i>Process:</i> External Processor	The External Processor submits a request status update through EWOC's OrderStatusUpdate Port.
Return Product Request Acknowledgement	One per status update request	<i>Process:</i> OrderFulfillment Port  <i>Classes:</i> OrderFulfillmentPortBindingImpl	<i>Process:</i> External Processor	The External Processor returns SubmitAcknowledgement which contains the information regarding the success of order submission to the External Processor.
Request Subsetted Product	One per product request	<i>Process:</i> OrderFulfillment Port	<i>Process:</i> EcDmEwoc  <i>Class:</i> EPDataAccessImpl	The EcDmEwoc places a request at External Processor for the granule to be subsetted.
Return Submit Acknowledgement	One per product request	<i>Process:</i> OrderFulfillment Port  <i>Class:</i> OrderFulfillmentProtBindingImpl	<i>Process:</i> EcDmEwoc  <i>Class:</i> OrderFulfillmentPortBindingImpl	The EWOC returns SubmitAcknowledgement which contains the information regarding the success of order submission from the External Processor.
Return Status Update Acknowledgement	One per status update request	<i>Process:</i> OrderStatusUpdate Port  <i>Class:</i> OrderServiceImpl	<i>Process:</i> EcDmEwoc  <i>Class:</i> OrderFulfillmentPortBindingImpl	The EWOC returns UpdateStatus with information regarding the success of request status update

**Table 4.4-4. EWOC CSCI Process Interface Events (3 of 3)**

Event	Event Frequency	Interface	Initiated By	Event Description
Login and logout	One every polling cycle	<i>Process:</i> AuthenticationService  <i>Class:</i> AuthenticationServicePortImpl	<i>Process:</i> EcDmEwoc  <i>Class:</i> CloseRequestHandlerImpl	The EWOC tries to login to ECHO's Authentication Service and receive a security token.
Return Submit Acknowledgement	One per product request	<i>Process:</i> OrderFulfillment Port  <i>Class:</i> OrderFulfillmentProtBindingImpl	<i>Process:</i> EcDmEwoc  <i>Class:</i> OrderFulfillmentPortBindingImpl	The EWOC returns SubmitAcknowledgement which contains the information regarding the success of order submission from the ECHO.
Close Provider Order	One every polling cycle	<i>Process:</i> OrderProcessingService  <i>Class:</i> OrderProcessingServicePortImpl	<i>Process:</i> EcDmEwoc  <i>Class:</i> CloseRequestHandlerImpl	The EWOC tries to update the status of order at ECHO for orders that are in terminal states.

#### 4.4.1.6 ECHO WSDL Order Component CSCI Data Stores

The EWOC CSCI calls OMS stored procedures to access OMS DB. The EWOC will have an OMS database interface only; it will not have an interface with the MSS database. The creation of MSS order and request objects will be handled through OMS stored procedure calls. Table 4.4-5 provides descriptions of the data stores shown in the EWOC CSCI architecture diagram.

**Table 4.4-5. ECHO WSDL Order Component CSCI Data Stores**

Data Store	Type	Functionality
OMS DB	PostgreSQL	OMS Database is designed to store the persistent information of user request, processing mode configuration, media configuration, staging policy configuration, Ftp Push policy configuration, request aging configuration, and request cleanup configuration.

#### 4.4.2 Data Management Subsystem Hardware

The primary components of the Data Management Subsystem include one hardware CI, Data Management Hardware CI (DMGHW). The general-purpose workstations are standalone hosts without fail-over capability. In the event of a host failure, any of the available workstations could be used to support end user DAAC maintenance.

#### **4.4.2.1 Data Management Hardware CI (DMGHW) Description**

The DMGHW CI includes Linux workstations. In the EBIS Document 920-TDx-001 (Hardware Design Diagram) provides descriptions of the HWCI, and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping. These workstations are used as end user workstations in maintenance of each of the respective DAAC sites.

## 4.5 Order Manager Subsystem Overview

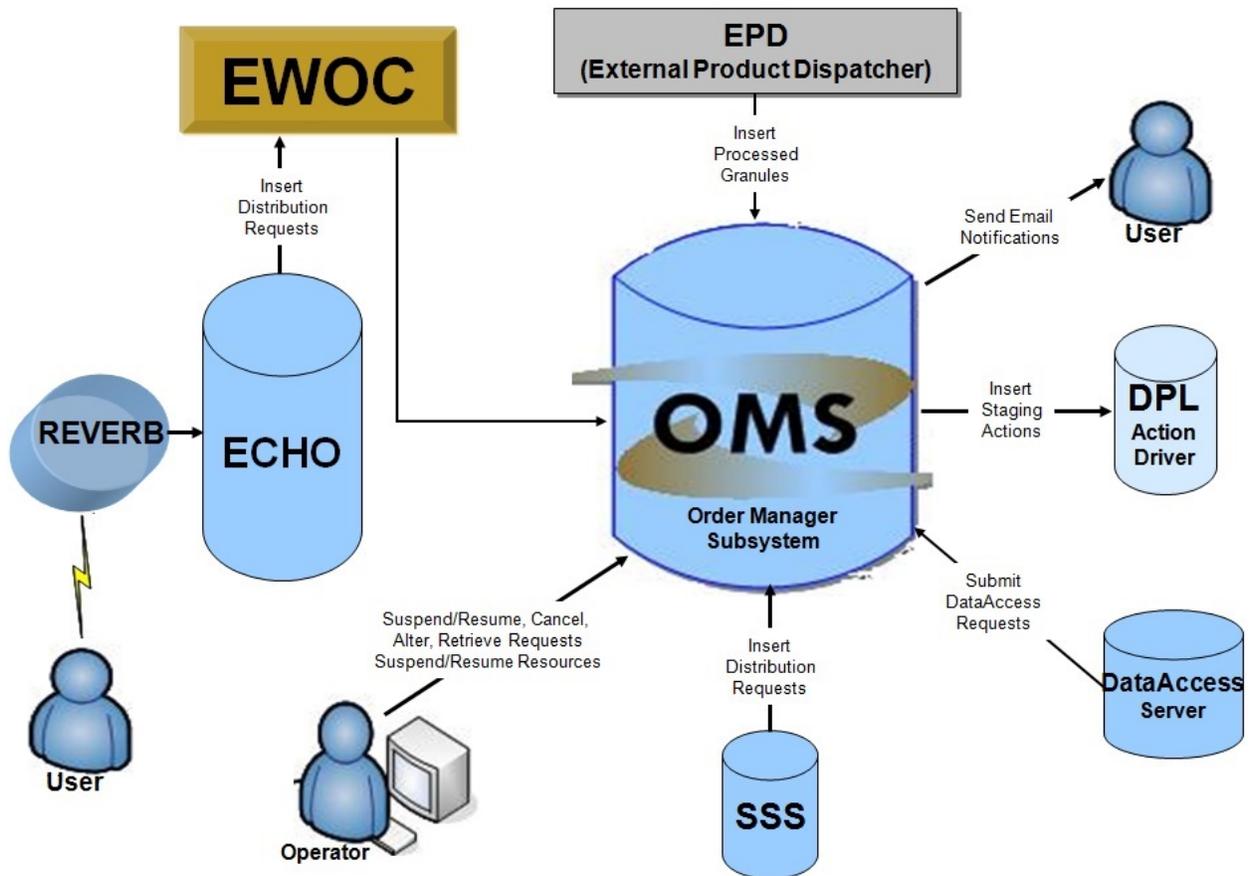
The Order Manager subsystem (OMS) manages all orders placed through ECHO via Reverb, the Spatial Subscription Server, and the EPD Server (i.e., external subsetter request and S4PM).

Once a request comes into the OMS subsystem, the server validates the request. Upon successful validation, the server stages the order in the Data Pool storage area. For Ftp Pull requests, links are created from the FtpPull directory to the files in the Data Pool storage. For Ftp Push requests, the Order Manager Server directly distributes the data. Upon successful shipment, OMS sends a Distribution Notice to the end user. An order is considered shipped as soon as the request status is updated to “Shipped”. For an FtpPull order, the request status is updated to “Shipped” after the file links are made in the Data Pool storage; for an FtpPush order, request status is “Shipped” after the Order Manager Server finishes pushing all the data associated with the order to its destination.

Special orders such as DataAccess (DA) and External Subsetter orders require further processing by the DA services or the External Subsetter. For DA orders, the Order Manager creates DA requests per granule based on the processing instructions in the original DA order. The DA requests are submitted to the DA service through the Reverb/EWOC workflow. The DA service processes the DA requests and returns the final output to the Order Manager Server which then distributes the final output to the end user. For External Subsetter Orders, the External Subsetter creates output granules which are then associated with the Order by the EPD Server. These output granules are later distributed by the Order Manager Server. The Order Manager Subsystem also includes a database that stores all order information persistently as soon as an order is received by ECS and before its receipt is acknowledged. This allows operators to resubmit an order if it encounters errors downstream, and allows the Order Management Service to perform some up front checks on the order and generate an operator intervention for the operator to handle the error conditions.

### Order Manager Subsystem Context

Figure 4.5-1 is the Order Manager Subsystem context diagram. The diagram shows the events sent to the Order Manager Subsystem and the events the Order Manager Subsystem sends to other subsystems. Table 4.5-1 provides descriptions of the interface events shown in the Order Manager Subsystem context diagram.



**Figure 4.5-1. Order Manager Subsystem Context Diagram**

**Table 4.5-1. Order Manager Subsystem Interface Events (1 of 2)**

Event	Interface Event Description
Insert Distribution Requests	The OMS receives Distribution Requests from the <b>Spatial Subscription Server (SSS)</b> , the OmSCLI and EWOC.
Suspend/Resume, Cancel, Alter and Retrieve Requests	The <b>Operator</b> suspends, resumes, cancels, alters and retrieves requests from the OMS (OMS DB).
Suspend/Resume Resources	The Operator suspends and resumes dispatching to all or selected resources.

**Table 4.5-1. Order Manager Subsystem Interface Events (2 of 2)**

Event	Interface Event Description
Send Email Notification	The OMS sends email notification to the requesting <b>user</b> when a request is altered, canceled or shipped or when a request enters operator intervention, or when an alert or intervention is generated.
Submit DA Request	The OMS submits DA requests to the DA Service that can perform the requested processing.

### Order Manager Subsystem Structure

- The Order Manager Subsystem consists of two CSCIs: the OMSRV and the OM GUI (described in the 609 document). The Order Manager Server (EcOmOrderManager) is a software configuration item. The Order Manager Server receives product distribution requests and submits them to the appropriate ECS component based upon the media type specified for the request. The OMS server stages granules associated with a request in the DPL storage area and then distributes the data via electronic media (i.e., Ftp Push, Ftp Pull, SCP). For special orders such as DAOrders, the Order Manager creates DA requests based on processing instructions and submits the DA requests to the DA service responsible for the type of processing requested. The output of a DA request is distributed to the end user. Similar to DA Orders, output granules associated with an External Subsetter request are distributed to the end user after being processed by the external subsetter. Order Manager Subsystem information is stored persistently in a relational Database Management System (DBMS). The Order Manager GUI (OMGUI) is used to monitor and control the operations of the Order Manager Server. In addition, the OMGUI is used to respond to Operator Intervention Requests generated by the Order Manager Server.

### Use of COTS in the Order Manager Subsystem

- RogueWave's Tools.h++  
The Tools.h++ class libraries are used by the OMS to provide basic functions and objects such as strings and collections. These libraries must be installed with the OMS software for any of the OMS processes to run.
- PostgreSQL libpq  
The PostgreSQL libpq provides access between OMS custom code and the PostgreSQL Server DBMS.
- PostgreSQL Server  
The PostgreSQL server provides access for OMS to insert, update and delete Distribution Requests, OMS configurations, and Operator Interventions. The PostgreSQL Server must be running during operations for the OMS to process Product Distribution Requests.

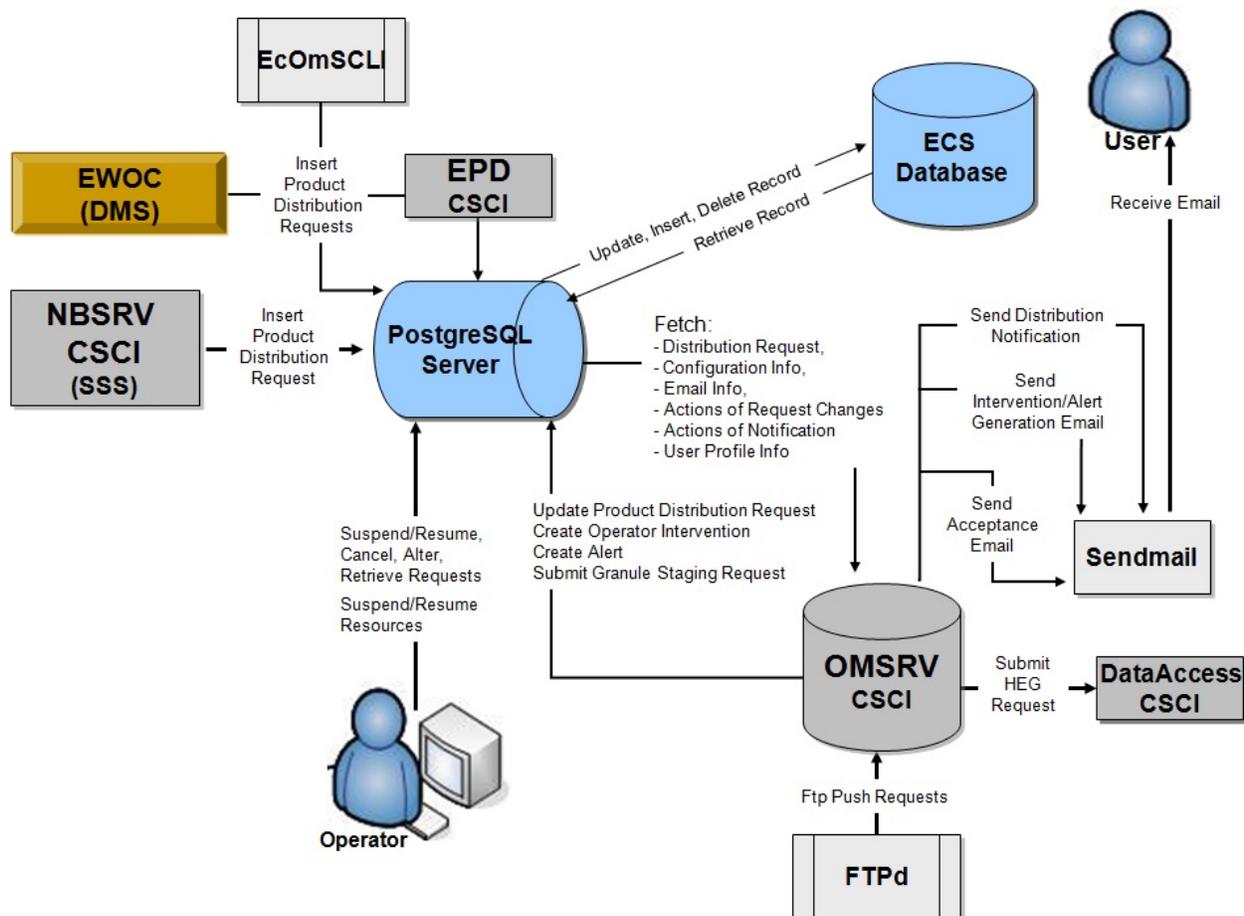
## **4.5.1 Order Manager Subsystem Software Description**

### **4.5.1.1 Order Manager Server CSCI Functional Overview**

The Order Manager Server (OMSRV) CSCI executes as a process and interacts with the Order Manager Database and the Data Pool System (DPL) Database. The Spatial Subscription Server (SSS), EWOC, and the OmSCLI (Command Line Interface) can be used to submit distribution requests to the OMS. These requests are all validated. Upon successful validation, the server stages the granules in a request in the DPL storage area. Electronic Ftp push media requests are directly pushed to the end user. Note that Order Manager Server dispatches Data Access (DA) requests to the DA service for processing before being distributed to the end user. For invalid requests, an Operator Intervention is generated. DAAC OPS personnel can use the Order Manager GUI to correct and resubmit the request. In response to an intervention, the Operator can also generate an email message, which is sent to the user by the Order Manager Server. The Order Manager Server also generates an alert and sends an email to a pre-configured email address when it detects internal or external resource failure. While a resource is suspended, the OMS Server halts dispatching of the requests that are utilizing the suspended resource.

### **4.5.1.2 Order Manager Server CSCI Context**

Figure 4.5-2 is the Order Manager Server CSCI context diagram. The diagram shows the events sent to the Order Manager Server CSCI and the events the Order Manager Server CSCI sends to other CSCIs. Table 4.5-2 provides descriptions of the interface events shown in the Order Manager Server CSCI context diagrams.



**Figure 4.5-2. Order Manager Server CSCI Context Diagram**

**Table 4.5-2. Order Manager Server CSCI Interface Events (1 of 2)**

Event	Interface Event Description
Send Distribution Notification	The OMSRV CSCI sends distribution notifications to the <b>end-users</b> .
Send Intervention/Alert Generation Email	The OMSRV CSCI sends intervention/alert generation email to the <b>end-users</b> .
Receive Email	<b>User</b> receives a status email sent by the OMSRV CSCI when request is put into Operator Intervention, is shipped or failed.

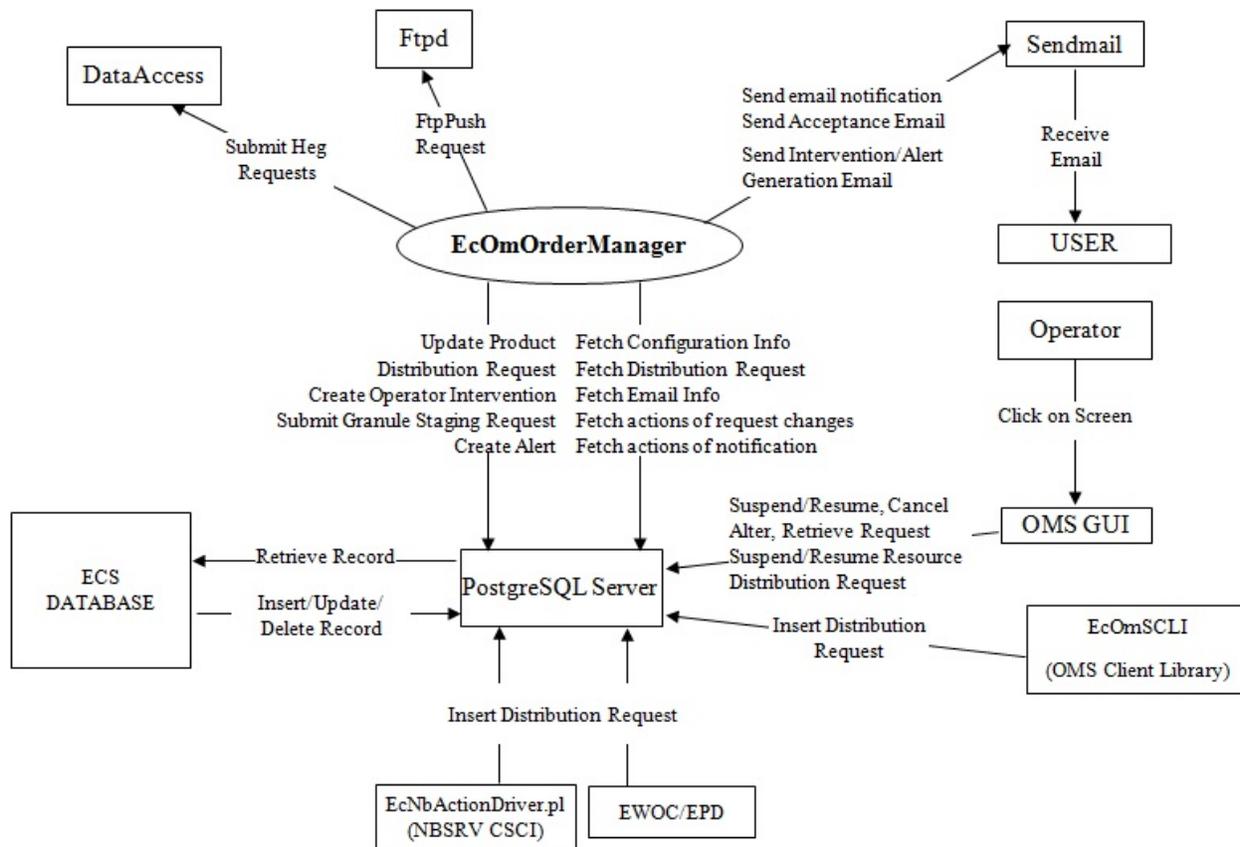
**Table 4.5-2. Order Manager Server CSCI Interface Events (2 of 2)**

<b>Event</b>	<b>Interface Event Description</b>
Insert Product Distribution Request	The NBSRV CSCI, SCLI CSCI, and DMS CSCI insert product distribution requests into the <b>PostgreSQL Server (OMS)</b> to be queued for processing by the OMSRV CSCI.
Insert Media Distribution Request	The DPL Web GUI inserts media distribution request into the <b>PostgreSQL Server (OMS)</b> .
Update Product Distribution Request	The OMSRV CSCI updates product distribution requests in the <b>PostgreSQL Server (OMS)</b> .
Create Operator Intervention	The OMSRV CSCI creates new Operator Intervention for request failures in the <b>PostgreSQL Server (OMS)</b> .
Create Alert	The OMSRV CSCI creates an alert for resource failures and stores the alert in the <b>PostgreSQL Server (OMS)</b> .
Submit granule staging request	The OMSRV CSCI submits a request to stage a granule to the <b>PostgreSQL Server (DPL storage in the OMS)</b> , which in turn calls DPL stored procedures to insert an action into the DPL DB.
Fetch Distribution Request	The OMSRV CSCI retrieves information associated with a product distribution request from the <b>PostgreSQL Server (OM)</b> .
Fetch Configuration Info	The OMSRV CSCI retrieves the OMSRV Configuration information from the <b>PostgreSQL Server</b> .
Fetch Email Info	The OMSRV CSCI retrieves information related to an operator intervention required to generate an email notification from the <b>PostgreSQL Server</b> .
Fetch actions of Request changes	The OMSRV CSCI retrieves actions regarding request changes, such as, request priority change, cancel request, suspend request, and update request ftppush parameters from the <b>PostgreSQL Server</b> .
Fetch actions of notification	The OMSRV CSCI retrieves actions regarding granule staged and DPL file system modified notification from the <b>PostgreSQL Server</b> .
FtpPush Requests	The OMSRV CSCI Ftp Pushes a request to the <b>end-user</b> .
Submit DA Request	The OMSRV CSCI submits DA requests to the DA Services for processing.
Suspend/Resume, Cancel, Alter and Retrieve Request	The Operator suspends, resumes, cancels, alters and retrieves requests from the <b>PostgreSQL Server (OMS DB)</b> .
Suspend/Resume Resources	The OMSRV CSCI suspends or resumes dispatching to all or selected resources in the <b>PostgreSQL Server</b> .
Update, Insert, Delete Record	The <b>PostgreSQL Server</b> performs update, insert, and delete database operations to the DB and DB.
Retrieve Record	The <b>PostgreSQL Server</b> performs retrieval database operations to/from the DB and DB.

#### 4.5.1.3 Order Manager Server CSCI Architecture

Figure 4.5-3 is the Order Manager Server (OMSRV) CSCI architecture diagram. The diagram shows the events sent to the OMSRV CSCI processes and the events the OMSRV CSCI processes send to other processes.

The OM Server CSCI consists of one process. This process is the EcOmOrderManager process.



**Figure 4.5-3. Order Manager Server CSCI Architecture Diagram**

#### 4.5.1.4 Order Manager Server CSCI Process Description

Table 4.5-3 provides descriptions of the processes shown in the OMSRV CSCI architecture diagram.

**Table 4.5-3. OMSRV CSCI Process**

Process	Type	Hardware CI	COTS/ Developed	Functionality
EcOmOrderManager	Server	OMSHW	Developed	The Order Manager Server prepares the request data in the DPL storage for distribution. FTP Push requests are directly pushed to the end user. Note that DA and external subsetter requests are first dispatched to the DA service or subsetter for processing. The processed output is then distributed to the end user. Order Manager Server sends a Distribution Notification to the end-user on completing an order.

EMD Baseline Information System (EBIS) Document 920-TDx-001 (Hardware Design Diagram) provides descriptions of the HWCI, and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

#### 4.5.1.5 Order Manager Server CSCI Interface Description

Table 4.5-4 provides descriptions of the interface events shown in the Order Manager Server (OMSRV) CSCI architecture diagram.

**Table 4.5-4. Order Manager Server CSCI Process Interface Events (1 of 4)**

Event	Event Frequency	Interface	Initiated By	Event Description
Insert Product Distribution Request	One per service request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EWOC, EPD <i>Script:</i> EcNbActionDriver.pl	EWOC, OmSCLand EcNbActionDriver.pl (SSS) insert product distribution request into OMS DB.
Submit DataAccessRequest	One or more per Service Request	<i>Process:</i> <i>DataAccess</i>	<i>Process:</i> EcOmOrderManager <i>Class:</i> OmSrDataAccessJob	The EcOmOrderManager submits DataAccessrequests to the service responsible for the requested processing.
Send acceptance email	One per email notification request	<i>Process:</i> Sendmail (COTS)	<i>Process:</i> EcOmOrderManager <i>Class:</i>	The EcOmOrderManager sends email notification to the end-users upon receipt of the request.
Send Intervention/ Alert Generation email	One per email notification request	<i>Process:</i> Sendmail (COTS)	<i>Process:</i> EcOmOrderManager <i>Class:</i>	The EcOmOrderManager sends intervention/ alert generation email to a configured user email account.
Receive Email	One per email notification	<i>End User:</i> specified in request	<i>Process:</i> Sendmail (COTS)	The User receives email sent by the EcOmOrderManager.
Send email notification	One per email notification request	<i>Process:</i> Sendmail (COTS)	<i>Process:</i> EcOmOrderManager <i>Class:</i> OmSrEmailRequest	The EcOmOrderManager sends email notifications to the end-users.

**Table 4.5-4. Order Manager Server CSCI Process Interface Events (2 of 4)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Fetch configuration info	One per startup	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Class:</i> OmSrDbInterface	The EcOmOrderManager retrieves configuration information from the PostgreSQL Server (OMS database).
Fetch Distribution Request	One per configurable interval	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Classes:</i> OmSrDbInterface, OmSrDistributionRequest	The EcOmOrderManager retrieves information associated with a product distribution request from the database.
Fetch actions of request changes	One per configurable interval	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Classes:</i> OmSrDbInterface, OmSrDistributionRequest	The EcOmOrderManager retrieves actions regarding request changes, such as, request priority change, cancel request, suspend request, and update request Ftp Push parameters.
Fetch actions of notification	One per configurable interval	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Classes:</i> OmSrDbInterface, OmSrDistributionRequest	The EcOmOrderManager retrieves actions regarding granule staged, and DPL file system modified notification.
Fetch email info	One per configurable interval	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Classes:</i> OmSrDbInterface, OmSrDistributionRequest, OmSrEmailRequest	The EcOmOrderManager retrieves information related to an operator intervention required to generate an email notification from the PostgreSQL Server (OMS).

**Table 4.5-4. Order Manager Server CSCI Process Interface Events (3 of 4)**

<b>Event</b>	<b>Event Frequency</b>	<b>Interface</b>	<b>Initiated By</b>	<b>Event Description</b>
Update Product Distribution Request	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Classes:</i> OmSrDbInterface, OmSrDistributionRequest	The EcOmOrderManager updates existing product distribution requests in the PostgreSQL Server (OMS).
Create Operator Intervention	One per request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Class:</i> OmSrDbInterface	The EcOmOrderManager creates a new Operator Intervention request in the PostgreSQL Server (OMS).
Create Alert	One per resource	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EcOmOrderManager <i>Library:</i> PostgreSQL libpq <i>Class:</i> OmSrDbInterface	The EcOmOrderManager creates an alert related to a resource failure such as Ftp Push Destination, Archive, and DPL File System failure to store in the PostgreSQL Server.
Insert Product Distribution Request	One per service request	<i>Process:</i> PostgreSQL Server (COTS)	<i>Process:</i> EWOC EPD EcOmCLI <i>Script:</i> EcNbActionDriver.pl <i>Library:</i> OmClientlib <i>Classes:</i> OmSrDbInterface	EWOC, EPDm the EcNbActionDriver, the EcOmSCLI, and the DPL Web Access GUI insert product distribution requests into the PostgreSQL Server (OMS).
Insert/Update/Delete/Retrieve Record	One per request	<i>Database:</i> OMS DB, DPL DB (COTS)	<i>Process:</i> PostgreSQL Server (COTS)	The PostgreSQL server performs database operations (inserts, updates, deletions and retrievals) to the OMS DB and DPL DB
Ftpush request	One per granule	<i>Process:</i> EcDIFtpService	<i>Process:</i> EcOmOrderManager	The EcOmOrderManager Ftp Pushes request data to the end-user.
Click on Screen	One per click	<i>Scripts:</i> OMS GUI scripts.	Operator	The Operator clicks on the screen to select the action.

**Table 4.5-4. Order Manager Server CSCI Process Interface Events (4 of 4)**

Event	Event Frequency	Interface	Initiated By	Event Description
Suspend/Resume, Cancel, Alter and Retrieve Requests	One per click	<i>Process:</i> PostgreSQL Server (COTS)	<i>Script:</i> OMS GUI script	The OMS GUI scripts send suspend/resume, cancel, alter and retrieve request commands to the PostgreSQL Server.
Suspend/Resume Resource	One per click	<i>Process:</i> PostgreSQL Server (COTS)	<i>Script:</i> OMS GUI script	The OMS GUI scripts send suspend/resume resource commands to the PostgreSQL Server.

#### 4.5.1.6 Data Stores

There are data stores associated with the Order Manager Server. They are the OMS DB, DPL DB. Table 4.5-5 provides a description of these data stores.

**Table 4.5-5. CSCI Data Stores**

Data Store	Type	Description
OMS	PostgreSQL	The ECS database is designed to store the persistent information of user request, processing mode configuration, media configuration, staging policy configuration, Ftp Push policy configuration, request aging configuration, and request cleanup configuration.
AIM	PostgreSQL	The ECS database implements the large majority of the persistent data requirements for the DPL subsystem which supports Large online cache of the majority of ECS data at each DAAC and avoids tape access to ECS archive.

## 4.6 Communications Subsystem Overview

The Communications Subsystem (CSS) provides the capability to:

- Transfer information internal to the Earth Observing System Data and Information System (EOSDIS) Evolution and Development (EED)
- Transfer information between the EED sites
- Provide connections between the ECS users and service providers
- Manage the ECS communications functions

### Communications Subsystem Context Diagram

Table 4.6-1 provides descriptions of the interface events shown in the CSS context diagrams.

**NOTE:** In Table 4.6-1 Request Communications Support is shown as a single event to simplify the table and provide a list of services available from CSS to the other CSMS subsystems.

**Table 4.6-1. Communications Subsystem (CSS) Interface Events**

Event	Interface Event Description
Request Communications Support (RCS)	<p>The CSS provides a library of services available to <b>each CSMS</b> subsystem. The subsystem services required to perform specific assignments are requested from the CSS. These services include:</p> <ul style="list-style-type: none"> <li>• CCS Middleware Support</li> <li>• Database Connection Services</li> <li>• Fault Handling Services</li> <li>• Mode Information</li> </ul>
Export Location Information	<p>The <b>OMS, DMS</b> send physical and logical server location information to the CSS for data location.</p>

### Communications Subsystem Structure

**Note:** The CSS logical names used in this document do not exactly match the physical names in the directory structure where the software is maintained. Therefore, after the logical name of each Computer Software Component (CSC) in parentheses, there is a physical directory structure name where the software is found. For example, the DCCI CSCI software can be found under the directory structure Distributed Object Framework (DOF) and the Server Request Framework software can be found under the directory structure /ecs/formal/common/CSCI\_SRF.

The CSS is composed of one CSCI, the Distributed Computing Configuration Item (DCCI, the software is found in directory DOF) and one HWCI. The CSS software is used to provide communication functions, processing capability, and storage.

## Use of COTS in the Communications Subsystem

**Note: The following RogueWave Libraries are currently delivered with custom code as static libraries. A separate installation of dynamic libraries is no longer required.**

- RogueWave's Tools.h++  
The Tools.h++ class libraries provide basic functions and objects such as strings and collections.
- RogueWave's DBTools.h++  
The DBTools.h++ C++ class libraries provide interaction, in an object-oriented manner, to the Sybase ASE database SQL server. The DBTools provide a buffer between the CSS processes and the relational database used.
- RogueWave's Net.h++  
The Net.h++ C++ class libraries, which provide functions and templates that facilitate writing applications, which communicate with other applications.

Other COTS products include:

- PostgreSQL RDBMS  
The PostgreSQL RDBMS provides persistent data storage for most EOSDIS components.
- CCS Middleware  
CSS middleware is no longer actively used by the servers but is built in to the binaries. Instead, the servers use direct host/port sockets to communicate with other servers.
- UNIX Network Services  
UNIX Network Services contain DNS, NFS, E-mail service, FTP, and TCP/IP capabilities.

## Error Handling and processing

EcUtStatus is a class used throughout the ECS custom code for general error reporting. It is almost always used as a return value for functions and allows detailed error codes to be passed back up function stacks.

When an error occurs, the error is logged into the applications log (ALOG). The Communications Subsystem (CSS) and System Management Subsystem (MSS) have two main mechanisms to handle the error:

1. Return an error status
2. Throw an exception.

The CSS uses the following classes for error handling and processing:

The EcUtStatus class is used to describe the operational status of many functions. The values most often reported are "failed" and "ok." But depending upon the application, detailed values could be set and sent. Please refer to the definition of this class (located in /ecs/formal/COMMON/CSCI\_Util/src/Logging/EcUtStatus.h) for all possible values.

The RWCString is used to store some status value returned by applications.

Integer is used to return some error status by applications. This is used specifically between client and server communications.

#### **4.6.1 The Distributed Computing Configuration Item Software Description**

The DCCI CSCI (the software is found in directory DOF) consists mainly of COTS software and hardware providing servers, gateways, and software library services to other CSMS CSCIs. The CSCI is composed of 17 computer software components (CSCs) briefly described here followed by a description of the HWCI.

The CSCI is composed of 17 computer software components (CSCs) briefly described here as processes followed by a description of the HWCI.

The remote file access group provides the capability to transfer and manage files using the following five functions: FTP, FTP Notification, Bulk Data Server (BDS), Network File System (NFS), and Filecopy.

1. The NFS (no physical directory) provides a distributed file sharing system among computers. NFS consists of a number of components, including a mounting protocol and server, a file locking protocol and server, and daemons that coordinate basic file service. A server exports (or shares) a filesystem when it makes the filesystem available for use by other machines in the network. An NFS client must explicitly mount a filesystem before using it.
2. The Filecopy utility (the software is found in directory /ecs/formal/common/CSCI\_Util/src/CopyProg) copies files from a specified source location to a specified destination location with options available for data compression. The DPL CSCI uses the Filecopy utility to transfer large files.
3. Virtual Terminal (no physical directory) provides the capability for the Operations staff on an ECS platform to remotely log onto another ECS machine.
4. Cryptographic Management Interface (CMI, the software is found in directory /ecs/formal/CSS/DOF/src/AUTHN) provides processes a means for obtaining random passwords and gaining access to Sybase.
6. The Infrastructure Library provides a set of services including the following.
  - Process Framework (PF) (the software is located in directory /ecs/formal/CSS/DOF/src/PF/pf): The PF is a software library of services, which provides a flexible mechanism (encapsulation) for the ECS Client and Server

applications to transparently include specific ECS infrastructure features from the library of services. (Library services include: process configuration and initialization, mode management and event handling, life cycle services (server start-up and shut-down), and set-up of security parameters.) The PF process is the encapsulation of an object with ECS infrastructure features and therefore the encapsulated object is fully equipped with the attributes needed to perform the activities assigned to it. The PF was developed for the ECS custom developed applications and is not meant for use by any COTS software applications. The PF ensures design and implementation consistency between the ECS Client and Server applications through encapsulation of the implementation details of the ECS infrastructure services. Encapsulation therefore removes, for example, the task of each programmer repeatedly writing common initialization code. The PF is built by first developing a process classification for the EED project from the client/server perspective. Then the required capabilities are allocated for each respective process level and type. PF-based ECS applications use Process Framework to read in their configuration information at startup. PF-based servers use Process Framework to initialize themselves as a CCS Middleware server and put it in a listen state to begin to accept requests from appropriate clients.

- Universal References (the software is found in directory `/ecs/formal/COMMON/CSCI_UR/src/UR/framework`): Universal References (URs) provide applications and users a system wide mechanism for referencing ECS data and service objects. Manipulating logical entities represented at run time as C++ objects in virtual memory performs ECS functions. Users and applications require references to the logical entities beyond the effective computational time to keep the objects in memory. Therefore, applications and users are given URs to these objects. Once an UR is made for an object, the object can be disposed of and later reconstituted from the UR. URs take up a small fraction of the space to keep in memory and can be externalized into an ASCII string, which an end user can manage. URs have the capability of re-accessing and/or reconstituting the object into memory as needed. Therefore, the object does not have to remain in memory, and can if appropriate, be written to a secondary storage system, like a database. While the UR mechanism guarantees reliable data externalization and internalization, the content of each type of UR is application specific. Only the object (this is referred to as the "UR Provider") that initially provides the UR is allowed to access and understand its content. URs are strongly typed to enforce appropriate access control to internal data both at compile time and during run time. Since URs are typed and have object specific data in them, separate UR object classes exist for each UR Provider class referred to. All of these UR classes use the mechanisms provided by the UR framework.
- Event Logging (the software is found in directory `LOGGING`): Event logging is the capability of recording events into files and provides a convenient way to generate and report detailed events. All ECS CSCIs use event and error logging as an audit trail for all transactions that occur during the ECS data processing and distributing.

- CSS software is executed on multiple hardware hosts throughout the ECS system to provide communication functions, processing capability, and storage. The software and hardware relationships are discussed in the CSS Hardware CI description.

#### 4.6.1.1 CCS Middleware Support Group Description

CSS middleware is no longer actively used by the servers but is built in to the binaries. Instead, the servers use direct host/port sockets to communicate with other servers

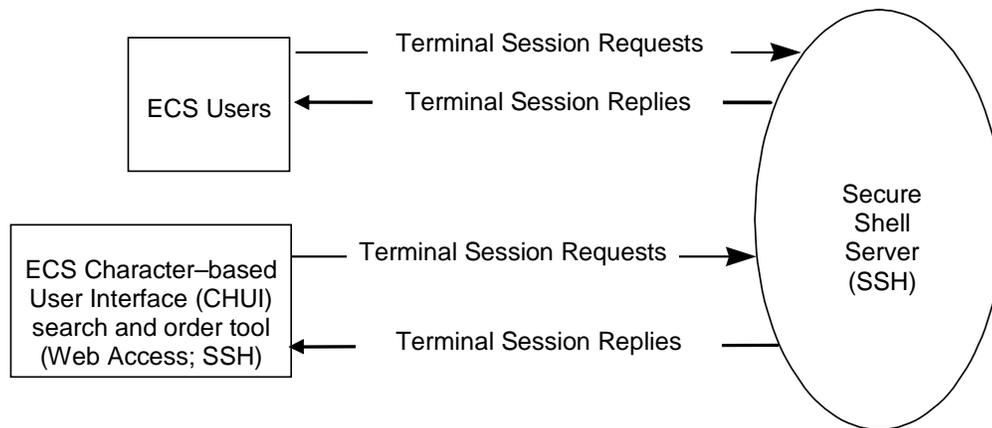
#### 4.6.1.2 Virtual Terminal Description

##### 4.6.1.2.1 Virtual Terminal Functional Overview

The Virtual Terminal (VT) effectively hides the terminal characteristics and data handling conventions from both the server host and Operations staff, and enables the Operations staff to remotely log on to other ECS machines. The CSS provides the secure shell server (sshd2) on available systems and common capability support for the ECS remote terminal service.

##### 4.6.1.2.2 Virtual Terminal Context

The CSS provides the secure shell (sshd2) remote access to the ECS systems. SSH is distributed as a third party remote server access service. The SSH service provides users with access to the ECS character-based user interface (CHUI) search and order tool. Figure 4.6-1 is the Virtual Terminal context diagram and Table 4.6-2 provides the descriptions of the interface events shown in the Virtual Terminal context diagram.



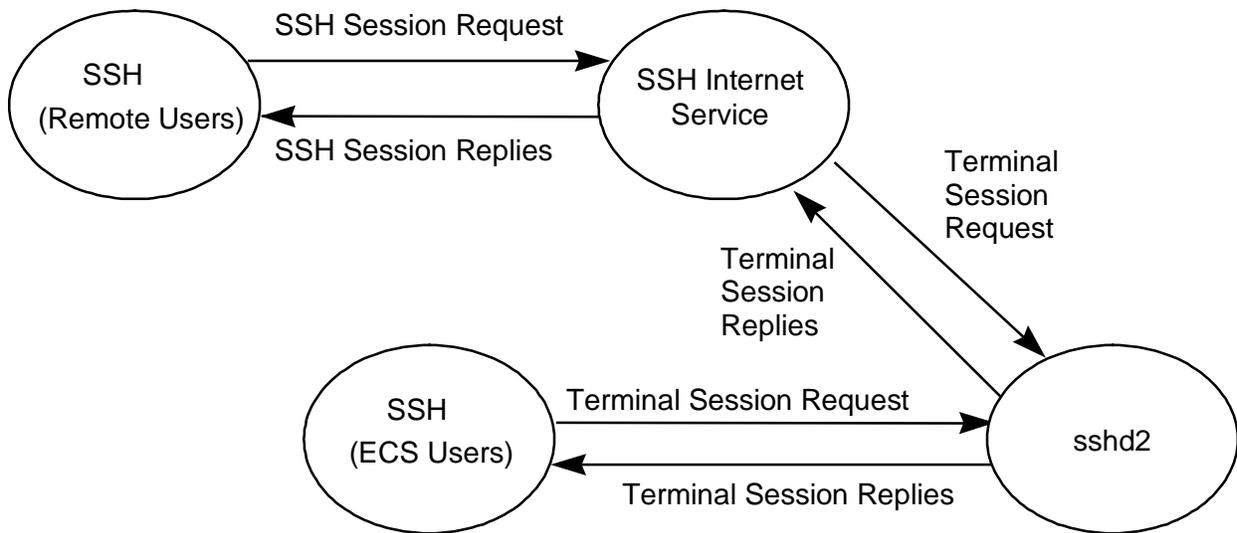
**Figure 4.6-1. Virtual Terminal Context Diagram**

**Table 4.6-2. Virtual Terminal Interface Events**

Event	Interface Event Description
Terminal Session Requests (Web access)	<b>ECS users</b> request a connection to a specified host via SSH.
Terminal Session Requests (ECS Users)	<b>ECS users</b> request a telnet session with a specified ECS host.
Terminal Session Replies (from SSH to ECS or other remote users)	The SSH Server residing on the ECS host responds to the terminal session requests and interacts via the successful connection.

**4.6.1.2.3 Virtual Terminal Architecture**

Figure 4.6-2 is the Virtual Terminal architecture diagram. The diagram shows the event traffic between the Remote Terminal Session with ECS Users and SSH with remote users.



**Figure 4.6-2. Virtual Terminal Architecture Diagram**

**4.6.1.2.4 Virtual Terminal Process Descriptions**

Table 4.6-3 provides the descriptions of the processes shown in the Virtual terminal architecture diagram.

**Table 4.6-3. Virtual Terminal Processes**

Process	Type	Hardware CI	COTS/ Developed	Functionality
SSH (Remote Users)	Client	CSSHW	COTS	Provides the dial-up terminal session as requested on the client-side via remote service.
SSH (ECS Users)	Client	CSSHW	COTS	Provides the user interface to a remote system using the SSH protocol.
(Internet Service)	Server/Client	CSSHW	COTS	Enables users to interact with the host through a remote service.
sshd2	Server	CSSHW	COTS	Function provides servers supporting SSH with virtual terminal protocol.

EBIS Document 920-TDx-001 (Hardware Design Diagram) provides descriptions of the HWCI, and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

#### 4.6.1.2.5 Virtual Terminal Process Interface Descriptions

Table 4.6-4 provides the descriptions of the interface events shown in the Virtual Terminal architecture diagram.

**Table 4.6-4. Virtual Terminal Process Interface Events**

Event	Event Frequency	Interface	Initiated by	Event Description
SSHSession Request	One per connection request	<i>Process:</i> sshd2 (dial-up service)	<i>Process:</i> SSH (COTS – remote users)	Any <b>ECS user</b> requiring a logon to another machine from the current machine. Users request to establish connection to a specified host via SSH.
SSH Session Replies	One per session reply	<i>Process:</i> SSH (remote service)	<i>Process:</i> sshd2 (remote service)	The <b>SSH Server</b> service provides <b>users</b> a remote session to request a terminal session to the secure shell client.
Terminal Session Request (SSH)	One per request to establish a session	<i>Process:</i> sshd2	<i>Process:</i> sshd2 (remote service)	Either the user or the client application service requests to establish a session with the specified host.
Terminal Session Replies (SSH)	One per connection request	<i>Process:</i> SSH (ECS users)	<i>Process:</i> sshd2	The Host Virtual Terminal Process, sshd2, responds to the connection requests and establishes or maintains the sessions.

#### 4.6.1.2.6 Virtual Terminal Data Stores

Data stores are not applicable for the Virtual Terminal.

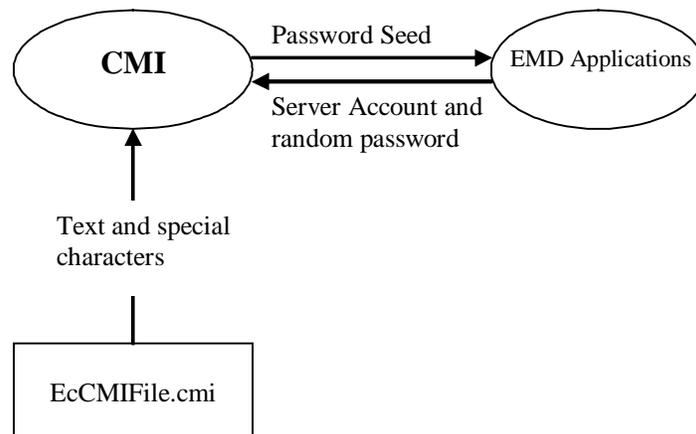
#### 4.6.1.3 Cryptographic Management Interface Software Description

##### 4.6.1.3.1 Cryptographic Management Interface Functional Overview

The Cryptographic Management Interface (CMI) classes provide the requesting process with a server account and a randomly generated password so the server can access security required services (i.e., Sybase ASE). These passwords (and optionally login names) are generated dynamically based on a psuedo-random number used as the seed for the password.

##### 4.6.1.3.2 Cryptographic Management Interface Context

Figure 4.6-3 is the Cryptographic Management Interface context diagram. Servers (PF or non-PF) use the CMI with a need for access to security required services. Table 4.6-5 provides descriptions of the interface events shown in the Cryptographic Management Interface context diagram.



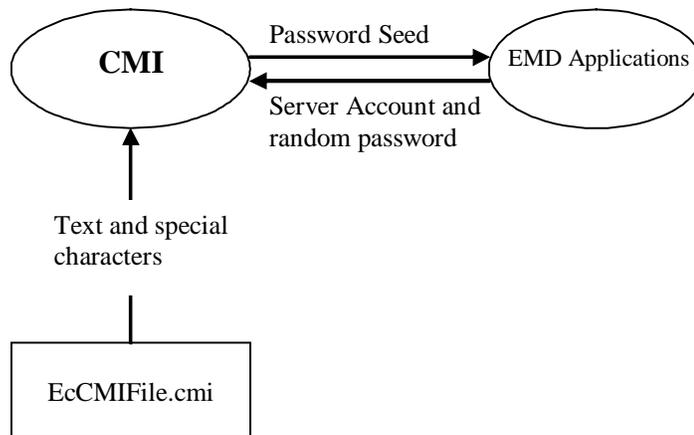
**Figure 4.6-3. Cryptographic Management Interface Context Diagram**

**Table 4.6-5. Cryptographic Management Interface Events**

Event	Interface Event Description
Password seed	The <b>ECS applications</b> request an account and provide a password seed to CMI.
Server account and random password	Account with random passwords created for the server is passed back to the server.
Text and special characters	Text and special characters read from a file for password generation.

### 4.6.1.3.3 Cryptographic Management Interface Architecture

Figure 4.6-4 is the Cryptographic Management Interface (CMI) architecture diagram. The diagram shows the event traffic between the CMI process and the ECS applications that interact with CMI for database connections.



**Figure 4.6-4. Cryptographic Management Interface Architecture Diagram**

### 4.6.1.3.4 Cryptographic Management Interface Process Descriptions

Table 4.6-6 provides descriptions of the processes shown in the Cryptographic Management Interface context diagram.

**Table 4.6-6. Cryptographic Management Interface Processes**

Process	Type	Hardware CI	COTS/ Developed	Functionality
ECS Process Names	Server	CSSHWS	Developed	Requests account with random password for access to security required services.
CMI	Other	CSSHWS	Developed	A server account and randomly generated password are returned to the requesting server.

EBIS Document 920-TDx-001 (Hardware Design Diagram) provides descriptions of the HWCI, and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

#### 4.6.1.3.5 Cryptographic Management Interface Process Interface Descriptions

Table 4.6-7 provides the descriptions of the interface events shown in the Cryptographic Management Interface architecture diagram.

**Table 4.6-7. Cryptographic Management Interface Process Interface Events**

Event	Event Frequency	Interface	Initiated by	Event Description
Password seed	One per password seed	<i>Process:</i> CMI <i>Library:</i> EcSeCmi <i>Class:</i> EcSeCmi	<i>DCCI Process:</i> EcCsRegistry	The server provides a unique number as a seed for generating a password to the <b>ECS Applications</b> .
Server Account and random password	One per account and password	<i>DCCI Process:</i> EcCsRegistry	<i>Process:</i> CMI <i>Library:</i> EcSeCmi <i>Class:</i> EcSeCmi	CMI generates a random password for the account based on the seed.

#### 4.6.1.3.6 Cryptographic Management Interface Data Stores

Table 4.6-8 provides descriptions of the data store shown in the Cryptographic Management Interface architecture diagram.

**Table 4.6-8. Cryptographic Management Interface Data Stores**

Data Store	Type	Functionality
EcCMIFile.cmi	File	This is a flat file of textual and special characters used by the CMI password generation algorithm to create passwords.

#### 4.6.1.4 Domain Name Server Software Description

##### 4.6.1.4.1 Domain Name Server Functional Overview

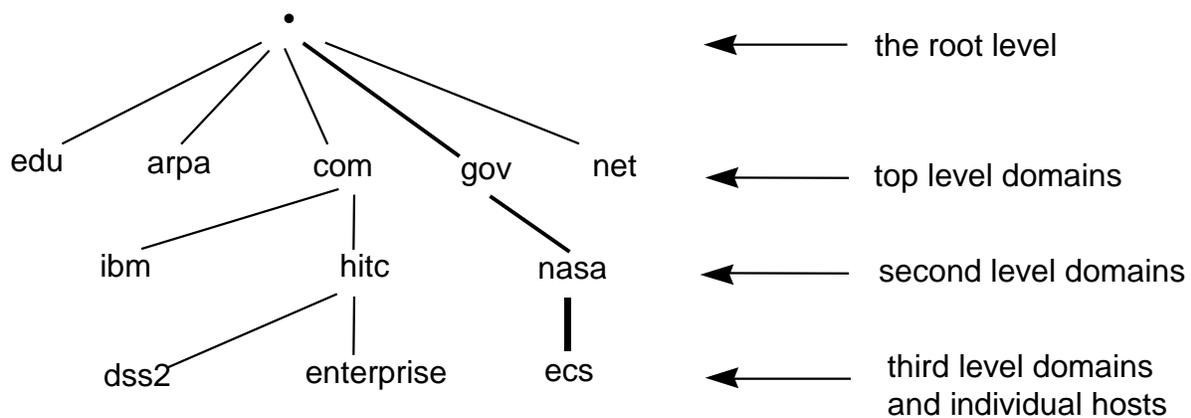
Domain Name Server (DNS) performs name-to-address and address-to-name resolution between hosts within the local administrative domain and across domain boundaries. DNS is COTS software implemented as server by running a daemon called “in.named.” Servers running the in.named daemon are referred to as name servers.

The server is implemented through a resolver instead of a daemon from the client side. The function of in.named is to resolve user queries for device names or addresses (DNS requires the address of at least one name server to be in the file /etc/resolv.conf). The name server, when

queried for a name or an address, returns the answer to the query or a referral to another name server to query for the answers.

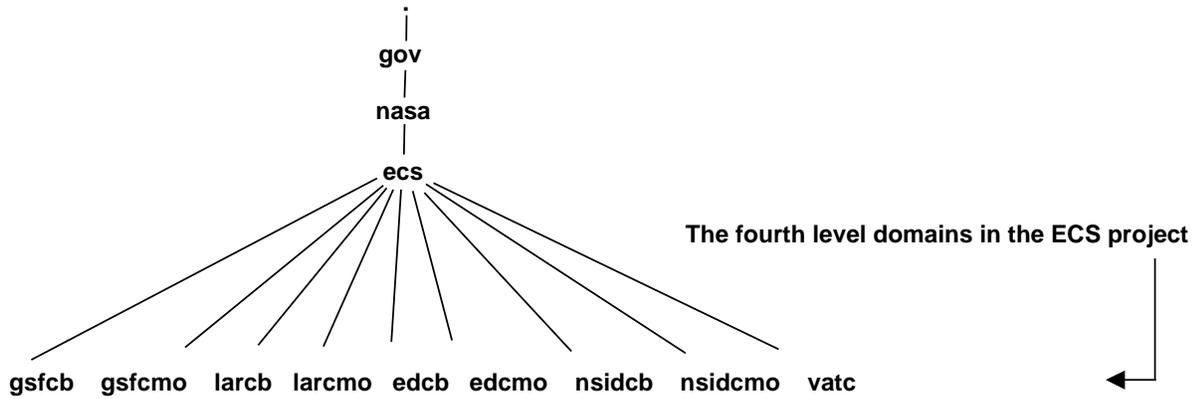
Each domain uses at least two kinds of DNS servers (primary and secondary) to maintain the name and address data corresponding to the domain. The primary server keeps the master copy of the data when it starts up in the “in.named,” daemon and delegates authority to other servers both inside and outside of its domain. A secondary server maintains a copy of the name and address data for the domain. When secondary server boots in.named, it requests the data for a given domain from the primary server. The secondary server then checks with the primary server periodically and requests updates to the daemon data so the secondary server is kept up to date with the primary.

DNS namespace is hierarchically organized, with nested domains, like directories. The DNS namespace consists of a tree of domains. Figure 4.6-5 is an Internet domain hierarchy diagram. The top-level domains are edu, arpa, com, gov, net, and for simplicity, not showing org, mil, and int, at the root level. The second level domain is nasa for gov. The third level domain is ecs for the EED project for nasa.gov.



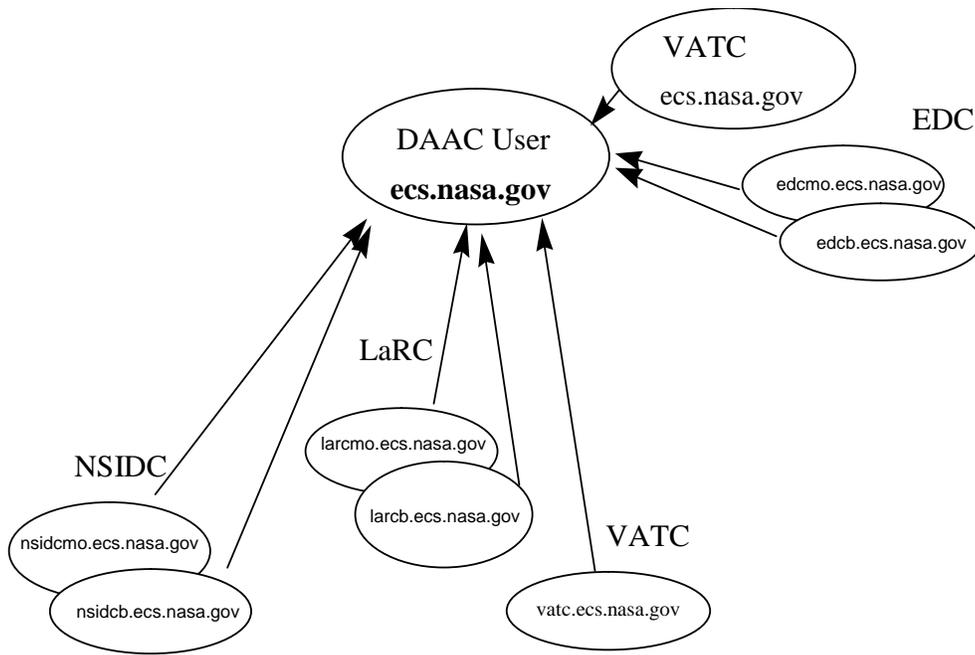
**Figure 4.6-5. Domains Hierarchy Diagram**

The fourth level domains in the EED project include domains of DAACs: gsfc, gsfcmo, and etc. Figure 4.6-6 is the hierarchy diagram of the fourth level domains in the EED project. The DAAC and M&O domains are part of the overall DNS. The top-level domain is ecs.nasa.gov and the two lower level domains for the DAACs, for example, gsfc.ecs.nasa.gov and gsfcmo.ecs.nasa.gov for the GSFC DAAC. The former is for the production network and the latter are for the GSFC M&O network.



**Figure 4.6-6. DNS Domains of the EED Project Diagram**

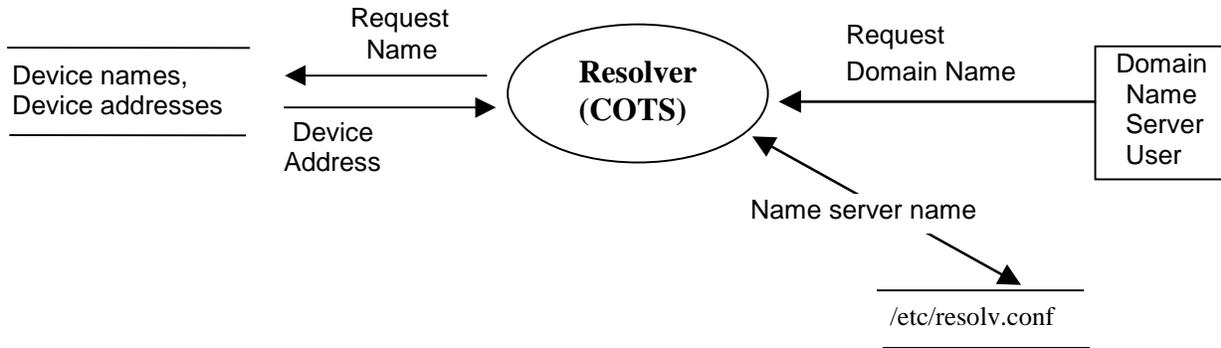
Figure 4.6-7 is the ECS topology domain diagram.



**Figure 4.6-7. ECS Topology Domains Diagram**

**4.6.1.4.2 Domain Name Server Context**

Figure 4.6-8 is the Domain Name Server context diagram.



**Figure 4.6-8. Domain Name Server Context Diagram**

#### 4.6.1.4.3 Domain Name Server Architecture

The Domain Name Server architecture diagram is the same as the context diagram and is not duplicated here. When the DNS client has a request for data, it searches the servers listed in the /etc/resolv.conf file in the order the servers were added to the file. When the first server does not contain the information of interest for the client, the second server in the list is searched and the search continues until the information is found.

#### 4.6.1.4.4 Domain Name Server Process Descriptions

Table 4.6-9 provides descriptions of the Domain Name Server processes shown in the Domain Name Server context diagram.

**Table 4.6-9. Domain Name Server Process**

Process	Type	Hardware CI	COTS/ Developed	Functionality
resolver	Client	CSSHW	COTS	Searches data store of device names and device addresses for information requested in the Domain Name Request. First entry in the file /etc/resolv.conf is used as the place to start searching.

EBIS Document 920-TDx-001 (Hardware Design Diagram) provides descriptions of the HWCI, and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

#### 4.6.1.4.5 Domain Name Server Process Interface Descriptions

Table 4.6-10 provides descriptions of the interface events shown in the Domain Name Server architecture diagram.

**Table 4.6-10. Domain Name Server Process Interface Events**

Event	Event Frequency	Interface	Initiated by	Event Description
Request Domain Name	One per user request	<i>COTS Software:</i> resolver	User	A DNS user requests data.
Name server name	One per search directory change	Data Store	<i>COTS Software:</i> resolver	The resolver retrieves the pathname for the directory to search for the user requested data from the /etc/resolv.conf database table. New file names are added to the list in the order they are stored.
Device Address	One per resolved address	<i>COTS Software:</i> resolver	<i>COTS Software:</i> name server	Returns the resolved address to the domain name requester via the Resolver.
Request Name	One per domain name request	<i>COTS Software:</i> name server	<i>COTS Software:</i> resolver	The resolver retrieves the domain name (device name and address) for the name server from an internal file used by the COTS software.

#### 4.6.1.4.6 Domain Name Server Data Stores

Table 4.6-11 provides descriptions of the data store shown in the Domain Name Server architecture diagram.

**Table 4.6-11. Domain Name Server Data Stores**

Data Store	Type	Functionality
/etc/resolv.conf	Other	Stores the primary and secondary server names.

#### 4.6.1.5 Infrastructure Libraries Group Description

##### 4.6.1.5.1 Infrastructure Libraries Group Functional Overview

The Infrastructure Library Group (ILG) is a library of reusable software frameworks and infrastructures used by ECS servers configured as a distributed client-server system. Table 4.6-12 provides descriptions of the infrastructures in the ILG.

**Table 4.6-12. Infrastructure Libraries (1 of 2)**

Library	Description
Process Framework (PF)	The PF is a software library of services, which provides a flexible mechanism (encapsulation) for the ECS client and server applications to transparently include specific ECS infrastructure features from the library of services, such as mode management, error and event logging, life-cycle services, and the CCS Middleware Naming Service.

**Table 4.6-12. Infrastructure Libraries (2 of 2)**

Library	Description
Server Request Framework (SRF)	The SRF infrastructure provides the standard for ECS synchronous and asynchronous communications between ECS applications. SRF is used to provide the client-server communications between the DPL INGEST Request Manager and Granule Server. SRF provides enhanced CCS Middleware call message passing and persistent storage as a CSS support capability with the described features available by subsystem request.
Universal References (UR)	A Universal Reference provider object from C++ objects generates UR during their run time in virtual memory. The UR is a representation of the original object. URs can be transformed from an object to an ASCII representation and again returned to an object. URs are objects the users and applications use with full capabilities. Once the UR is obtained, the original object can be discarded and later reconstituted and used. URs can refer to objects local or remote to an address space. Therefore, the object does not have to remain in memory, and can, as appropriate, be written to a secondary storage system like a database.
Error/Event Logging	Event/Error logging is the capability of recording events into files and provides a convenient way to generate and report detailed events. All ECS CSCIs use event and error logging as an audit trail for all transactions (requests for data or services) that occur during the ECS data processing and distributing.
Message Passing (MP)	Message Passing provides peer-to-peer asynchronous communications service, which notifies clients of specific event triggers. This service is provided upon subsystem request by the CSS. It is an alternative means of communication.
ServerUR	Provides unique identification (universal reference) for data and service objects in the ECS. The Server Locator is a class that enables servers to register their location without referring to its physical location and be uniquely identified and located in the ECS. Client applications use the Server Locator to find any registered server. The Server Locator is used in ECS in any client-server CCS Middleware-based communication.
Fault Handling (FH)	The Failure Recovery Framework provides a general-purpose fault recovery routine enabling client applications to reconnect with servers after the initial connection is lost. This is accomplished through the CCS Naming Service, through which the Failure Recovery Framework can determine whether a server is listening. The Failure Recovery Framework provides a default and configurable amount of retries and duration between retries. This fault recovery takes effect for each attempt by the client to communicate with the server for all applications that employ the Failure Recovery Framework.
DBWrapper directory	The DBWrapper directory is the DBMS Interface Infrastructure Library used by ECS applications to connect to the Sybase ASEs. Sybase ASEs operate by ECS defined guidelines for mode management, thread safety, error handling, error recovery, security, configuration management, and performance of database connections.

#### 4.6.1.5.2 Infrastructure Libraries Group Context

A context diagram is not applicable to the Infrastructure Libraries Group.

#### 4.6.1.5.3 Infrastructure Libraries Group Architecture

An architecture diagram is not applicable to the Infrastructure Libraries Group.

#### 4.6.1.5.4 Infrastructure Libraries Group Process Descriptions

Descriptions of the individual processes in the Infrastructure Libraries Group are not applicable.

#### 4.6.1.5.5 Infrastructure Libraries Group Interface Descriptions

Table 4.6-13 provides descriptions of the interfaces the Infrastructure Libraries Group.

**Table 4.6-13. Infrastructure Libraries Group Interfaces (1 of 3)**

Library	Interface	Initiated by	Library Description
Process Framework (PF)	<i>Library:</i> EcPf <i>Classes:</i> EcPfManagedServer, EcPfClient	EcCsRegistry, DPL Servers OMS	The PF is a software library of services, which provides a flexible mechanism (encapsulation) for the ECS client and server applications to transparently include specific ECS infrastructure features from the library of services. Features and services include: <ul style="list-style-type: none"><li>• Mode management</li><li>• Error and event logging</li><li>• Life-cycle services</li><li>• CCS Naming Service</li></ul>

**Table 4.6-13. Infrastructure Libraries Group Interfaces (2 of 3)**

Library	Interface	Initiated by	Library Description
Universal References (UR)	<i>Library (Common):</i> EcUr	Object Origination	A Universal Reference provider object from C++ objects generates UR during their run time in virtual memory. The UR is a representation of the original object. URs can be transformed from an object to an ASCII representation and again returned to an object. URs are objects the users and applications use with full capabilities. Once the UR is obtained, the original object can be discarded and later reconstituted and used. URs can refer to objects local or remote to an address space. Therefore, the object does not have to remain in memory, and can, as appropriate, be written to a secondary storage system like a database.
Error/Event Logging	<i>Library:</i> event <i>Class:</i> EcLgErrorMsg	DPL Servers OMS	Event/Error logging is the capability of recording events into files and provides a convenient way to generate and report detailed events. All ECS CSCIs use event and error logging as an audit trail for all transactions (requests for data or services) that occur during the ECS data processing and distributing.
ServerUR	<i>Library (Common):</i> EcUr <i>Class:</i> EcUrServerUR	<i>Processes:</i> EcOmOrderManager DPL Servers <i>Classes:</i> EcNsServiceLoc  <i>DSS Libraries:</i> DsBt, DsDe1, DsGe	Provides unique identification (universal reference) for data and service objects in the ECS. The Server Locator is a class that enables servers to register their location without referring to its physical location and be uniquely identified and located in the ECS. Client applications use the Server Locator to find any registered server. The Server Locator is used in ECS in any client-server CCS Middleware-based communication.

**Table 4.6-13. Infrastructure Libraries Group Interfaces (3 of 3)**

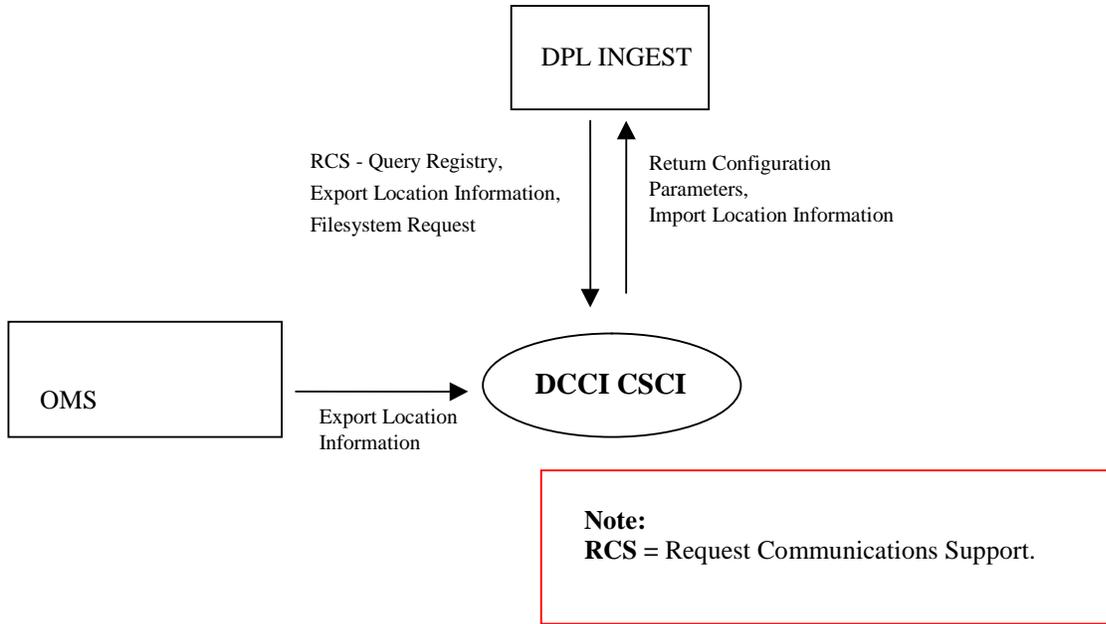
Library	Interface	Initiated by	Library Description
Fault Handling (FH)	<i>Library:</i> EcFh  <i>Class:</i> EcFhExecutor	EcOmOrderManager DPL Servers	The Failure Recovery Framework provides a general-purpose fault recovery routine enabling client applications to reconnect with servers after the initial connection is lost. This is accomplished through the CCS Naming Service, through which the Failure Recovery Framework can determine whether a server is listening. The Failure Recovery Framework provides a default and configurable amount of retries and duration between retries. This fault recovery takes effect for each attempt by the client to communicate with the server for all applications that employ the Failure Recovery Framework.

**4.6.1.5.6 Infrastructure Library Group Data Stores**

Data Stores are not applicable for the Infrastructure Library Group.

**4.6.2 The Distributed Computing Configuration Item Context**

Figure 4.6-9 is the Distributed Computing Configuration Item (DCCI) CSCI context diagrams. The diagrams show the events sent to the DCCI CSCI and the events the DCCI CSCI sends to other CSCIs. Table 4.6-14 provides descriptions of the interface events shown in the DCCI CSCI context diagrams.



**Figure 4.6-9. Distributed Computing Configuration Item (DCCI) CSCI Context Diagram**

**Table 4.6-14. Distributed Computing Configuration Item (DCCI) CSCI Interface Events (1 of 2)**

Event	Interface Event Description
Filesystem Request	The NFS clients request ECS files or directories via an established mount point. The NFS Server makes the storage device(s) and its data accessible for use by the clients.
Submit Subscription	The <b>DPL CSCI</b> submits a subscription request to the DCCI CSCI using the advertisement subscribing to an insert event for an ESDT.
Password Seed	The <b>DPL CSCI</b> requests an account and provides a password seed to the CMI.
Notify of Subscription	The DCCI CSCI sends notification (via message passing) to the <b>DPL CSCI</b> when the subscribed event occurs.
Server Account and random password	An account with random passwords, created for the server, is passed back to the server in the <b>DPL CSCI</b> from the DCCI CSCI.
Password Seed	The <b>DPL CSCI</b> requests an account and provides a password seed to the CMI.
Server Account and random password	An account with random passwords, created for the server, is passed back to the server in the <b>DPL CSCI</b> from the DCCI CSCI.

**Table 4.6-14. Distributed Computing Configuration Item (DCCI) CSCI Interface Events (2 of 2)**

Event	Interface Event Description
Request Communications Support	<p>The DCCI CSCI provides a library of services available to the <b>DPL INGEST CSCI</b>. The CSCI services required to perform specific assignments are requested from the DCCI CSCI. These services include:</p> <ul style="list-style-type: none"> <li>• Database Connection Services</li> <li>• File Transfer Services</li> <li>• Network &amp; Distributed File Services</li> <li>• Bulk Data Transfer Services</li> <li>• File Copying Services</li> <li>• Name/Address Services</li> <li>• Password Services</li> <li>• Server Request Framework (SRF)</li> <li>• Universal Reference (UR)</li> <li>• Error/Event Logging</li> <li>• Message Passing</li> <li>• Fault Handling Services</li> <li>• Mode Information</li> </ul>

### 4.6.3 Distributed Computing Configuration Item Architecture

An architecture diagram is not applicable for the DCCI CSCI. However, Table 4.6-15 shows the mapping between CSMS CSCIs and CSS CSCs.

**Table 4.6-15. CSMS CSCI to CSS CSC Mappings**

CSMS CSCI	CSS CSC	Process Used	CSS Libraries Used
DPL INGEST	<ul style="list-style-type: none"> <li>- E-Mail Parser Gateway Server</li> <li>- FTP</li> <li>- NFS</li> </ul>	<ul style="list-style-type: none"> <li>- EcCsEmailParser</li> <li>- ftp_popen</li> <li>- NFS Client</li> </ul>	<ul style="list-style-type: none"> <li>- PF</li> <li>- ServerUR</li> <li>- Error Logging</li> <li>- Event Logging</li> <li>- UR</li> <li>- Fault Handling Services</li> <li>- Server Request Framework (SRF)</li> </ul>

### 4.6.4 Distributed Computing Configuration Item Process Descriptions

Process descriptions are not applicable for the DCCI CSCI.

### 4.6.5 Distributed Computing Configuration Item Process Interface Descriptions

Process interface descriptions are not applicable for the DCCI CSCI.

#### **4.6.6 Distributed Computing Configuration Item Data Stores**

Data stores are not applicable for the DCCI CSCI.

#### **4.6.7 Communications Subsystem Hardware CI Description**

Document 920-TDx-001 (HW Design Diagram) provides descriptions of the Distributed Computing Configuration HWCI and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

Three DCCI software programs run on this host including the Domain Name Server (DNS), Network Information Services (NIS), and Mail Server. DNS enables host names to be distinguished based on their host name and IP address relationship. NIS is a service that stores information that users, workstations, and applications must have to communicate across the network. This information includes machine addresses, user names, passwords, and network access permissions. The Mail Server provides standard electronic mail capability.

The CSS Server is a stand-alone host and intrinsically does not have fail-over capability. DNS and Distributed Time Service (DTS) are loaded on multiple hosts designated as secondary. Any one of these hosts can operate as primary servers for the DNS or DTS services in the event of non-recoverable hardware failure of the primary host.

## **4.7 Internetworking Subsystem (ISS) Overview**

The Internetworking Subsystem (ISS) contains one hardware configuration item (HWCI), the Internetworking HWCI. INCI provides internetworking services based on protocols and standards corresponding to the lower four layers of the OSI reference model as described below.

### **Transport Protocols**

EED provides IP-based connection-oriented and connectionless transport services. The connection-oriented service is implemented using TCP, while User Datagram Protocol (UDP) is used for connectionless transport. Higher layer applications use one or the other based on such requirements as performance and reliability.

Transmission Control Protocol (TCP), specified in RFC 793, is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols to support multi-network applications. It provides for reliable inter-process communication between pairs of processes in host computers attached to networks within and outside EED. Because TCP assumes it may obtain potentially unreliable datagram service from the lower level protocols, it involves additional overhead due to the implementation of re-transmission and acknowledgment processes.

The UDP, specified in RFC 768, provides a procedure for application programs to send messages to other programs with minimal overhead. The protocol is transaction oriented and delivery of data is not guaranteed, since there is no acknowledgment process or re-transmission mechanism. Therefore, applications requiring ordered and reliable delivery of data would use TCP.

EED Public Services are available with IPv6 as mandated by NASA EOSDIS requirement.

### **Network Layer Protocols**

The network layer provides the functional and procedural means to transparently exchange network data units between transport entities over network connections, both for connection-mode and connectionless-mode communications. It relieves the transport layer from concern of all routing and relay operations associated with network connections.

The Internet protocol (IP) Version 4, specified in RFC 791, is the EED supported network protocol, based on its dominance in industry usage and wide community support. As part of IP support, ICMP and ARP are also supported.

### **Physical/Datalink Protocols**

Physical and data-link protocols describe the procedural and functional means of accessing a particular network topology. For the DAAC networks, the data-link/physical protocol is 1000 Gb/s and 10Gb/s. Ethernet with LP DAAC added 10Gbps Ethernet to their network to their HighSpeed DMZ network.

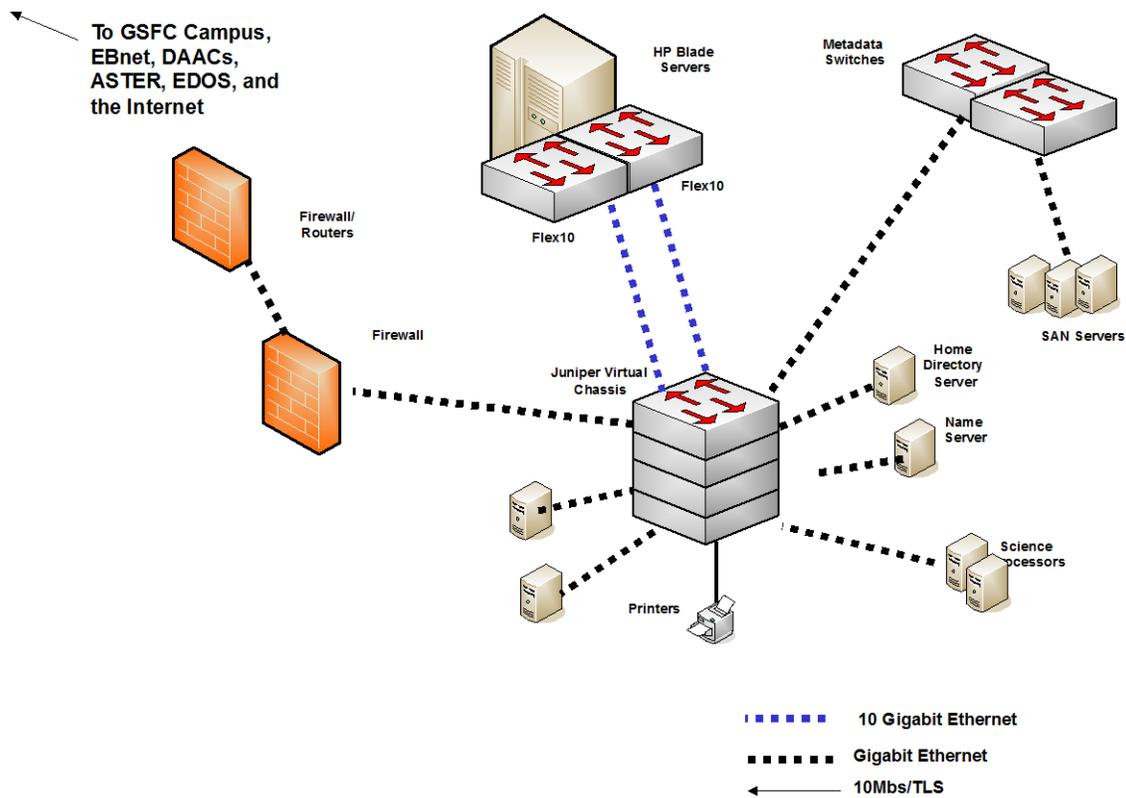
## 4.7.1 Internetworking Subsystem Description

### 4.7.1.1 DAAC LAN Architecture

This section provides an overview of the DAAC network architecture. Information on DAAC specific implementation level detailed designs can be found in Section 4.7.1.4.

The generic architecture for DAAC Local Area Networks (LANs) is illustrated in Figure 4.7-1. The topology consists of a Production Network, and a SAN LAN Network. The Juniper SRX650 protects the production network at NSIDC. The general network layout is displayed in the diagram below.

Note that not all sites have the complete complement of hardware and subsystems shown in Figure 4.7-1. For instance EDC and LaRC do not have EED managed routers or firewalls, their networks are protected by local perimeter campus firewalls at their site locations. The most recent changes on a network are introduction of Juniper Virtual Chassis as replacement for obsolete Cisco Core 6506E.



**Figure 4.7-1. DAAC Networks: Generic Architecture Diagram**

The Production Network consists of a Juniper VCs (four switches formed into virtual chassis). All servers, workstations, printers, and the BladeCenter chassis are connected to individual switch ports.

The SAN Metadata network consists of a Catalyst 3560 or 3750x multi-port Ethernet Switches. This network is used for the StorNext file system MetaData and to manage the storage arrays. All servers which use the StorNext file system and storage arrays are connected to this network. The virtual servers in the HP chassis connect via the two internal 10G Flex10 virtual appliances to with redundant 10G Fiber uplinks. These uplinks are forwarding/tagging vlans for production, metadata and management purposes.

#### **4.7.1.2 DAAC Addressing and Routing Architecture**

Most of the devices connected to the Production Network are assigned Class C address space with the exception that LaRC, uses a /23 subnet for production network NSIDC devices connected to the Production Network and all devices that are connected to the SAN LAN are assigned private addresses as specified in RFC 1918 (as of 02/96). Documents that list IP address assignments to all hosts and network attached devices are listed in Table 4.7-1. All external EED address space (except for addresses used on the Metadata Ethernet networks) is provided from address blocks designated by EOSDIS Ebnet.

The internal routing between EDN, PVC, VATC and WebRail DMZ is performed via static routes on Juniper SRX650 Firewall. Site-to-Site IPsec VPN is the main gateway connecting EED networks to GSFC. The IP addresses assigned to all of the networks are from EOS RFC1918 IP address blocks. There is an outbound and inbound block on both sites of the tunnel. EED IPsec firewall and EISOC firewall. The tunnel is running over a 10 Mb/sec TLS link. There are no split tunnels allowed so all network connections appear to the outside world to come from GSFC. The Remote Access for telecom/mobile workers is provided via NASA EOS SSL Client VPN. User authentication is performed using Code700 key fobs.

#### **4.7.1.3 Network-based Security Architecture**

The network architecture provides a strong level of security by implementing stateful firewalls at the DAACs. The firewalls block incoming network traffic unless there is a rule specifically allowing the traffic to pass into the DAACs. Note: that in addition to network-based security; EED has implemented other security measures, such as two-factor authentication, secure shell (SSH) and host access lists (ACLs), which are discussed in the CSS sections of this document.

#### **4.7.1.4 Internetworking Subsystem Detailed Design**

The ISS implementation level detailed design is documented in the documents listed in Table 4.7-1. Document 920-TDx-001 (HW Design Diagram) provides descriptions of the ISS HWCI and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

All of the documents are under configuration control and can be obtained from EED Configuration Management (CM). The documents are not on line for security reasons. Therefore special authorization is needed for their release.

**Table 4.7-1. Internetworking Subsystem Baseline Documentation List**

Document Name	EDC	LaRC	NSIDC
Hardware/Network Diagram	921-TDE-002	921-TDL-002	921-TDN-002
Host IP Address Assignment Table	921-TDE-003	921-TDL-003	921-TDN-003
Network Hardware IP Address Assignment	921-TDE-004	921-TDL-004	921-TDN-004

#### 4.7.2 Internetworking Hardware HWCI (INCI)

This HWCI provides the networking hardware for internal and external DAAC connectivity. The HWCI includes Ethernet switches and cabling; routers and cabling; and network test equipment. Each network hardware device is discussed in detail below.

The DAAC LANs contain three types of COTS hardware: Firewall, Ethernet switches, and Routers. All hosts in the DAACs are attached to Ethernet switches. The Routers are used to provide access to external networks (NISN, Abilene, and Campus nets). Table 4.7-2 provides a list of networking hardware used in EED networks.

The following descriptions of Network Hardware devices are provided as illustrative detail. All details of the hardware configuration should be verified with the appropriate Hardware/Network documents listed in Table 4.7-1.

**Table 4.7-2. Networking Hardware for EED Networks**

Networking Hardware	Vendor
Firewall	Juniper SRX650
Router (EED Router)	Cisco 7204vvr
Ethernet Switch	Catalyst 3560G/3750x
Ethernet Switch	Juniper EX4550
Ethernet Switch	Juniper EX4200
Ethernet Cables	1000baseT connection to servers, workstations, printers, PCs, and x-terms

##### 4.7.2.1 EED Ethernet Switch

The EED Core Ethernet switch is the Juniper Virtual Chassis with multiple 1Gb/s and 10Gb/s ports and powerful packet engines. It forms the core of the EED Production network by interconnecting all servers, workstations, printers, PCs, and Sun Ray terminals. The switch has

redundant power supply and fan units. It also has redundant packet engines. All modules are hot swappable.

#### **4.7.2.2 EED Router**

The EED Router is a Cisco 7200 series router running Cisco's Internetwork Operating System (IOS). The router has three 1Gb/s Ethernet ports. The EED Router is only used at NSIDC and it provides connectivity to EED sites and the Internet via its interfaces with NISN and the local campus network.

The ECS Router has redundant power supply and fan units.

For support purposes, the VATC also have 7200 router.

#### **4.7.2.3 SAN LAN Ethernet Switch**

The EED Metadata Ethernet switch is a Cisco Catalyst 3560G switch capable of supporting up to 48 1000 Mbps ports.

#### **4.7.2.4 Firewall**

The EED Firewall is a Juniper SRX650 Services Gateway. It is a stateful type firewall, which is capable of supporting several 1 Gbps Ethernet interfaces with 7Gbps total throughput. The firewall has a special software build version to accommodate the large FTP traffic.

At EDC and LaRC, the Juniper Virtual Chassis interface directly with the Campus firewall. The Campus routers provide all external network connectivity.

## 4.8 EED General Process Failure Recovery Concepts

During Earth Observing System Data and Information System (EOSDIS) Evolution and Development (EED) processing, client or server failures can occur. These failures cause certain recovery events to take place within the EED. To understand the General Process Failure Recovery of the EED processes, several key concepts must be described. These failure recovery concepts are:

- 1) Client-Server Rebinding
- 2) Database Reconnecting
- 3) Request Identification
- 4) Request Responsibility
- 5) Queues
- 6) Request Responses
- 7) Duplicate Request Detection
- 8) Server Crash and Restart
- 9) Client Crash and Restart

These concepts compose the general philosophy of the EED process failure recovery. The General Process is performed as a “process chain” to service requests for data or other services (e.g., order tracking or data retrieval from another processing system) via the client/server architecture. A brief description of each of the key concepts for General Process Failure Recovery follows.

### 4.8.1 Client-Server Rebinding

EED performs communication between client and server through the database. A client will insert information concerning its request into the database and the server will poll the database to pick up new requests. This allows for persistent client/server messaging that is inherently fault tolerant. The exceptions to this are the EcDIQuickServer, HegServer, and EcDIftpService, as well as the Data Access components (such as EGI and HegService) which communicate with clients using sockets and HTTP connections. If a client of any of these services is unable to connect, it should suspend the resource associated with that service after a configurable number of retries.

### 4.8.2 Database Reconnecting

Communication with the PostgreSQL database relies on standard libraries specific to each programming language. A Perl program should use the perl DBI module interface, a Java process should use a JDBC interface. A custom class that wraps the PostgreSQL libpq client library, EcCmdb is used for C++ code. In each of these cases the calling program should

attempt a connection for a configurable number of times, waiting for a configurable amount of time between each connection attempt.

EED applications that require the establishment of a large number of connections should implement a connection pool that allows them to reuse connections. EED applications should handle deadlock errors with a configurable retry. If the request to the database succeeds before the retries are exhausted, the application continues normally and completes the transaction in progress when the Database fault occurred.

However, operations should be aware of the following facts:

- Not all EED applications are able to use the EED database interface code
- Not all EED applications are able to use database transactions and automatic re-connection in the manner described.

### **4.8.3 Request Identification**

EED identifies requests with primary keys stored in the database. The use of RPC ID is restricted to a small set of OMS clients including the acquire script which is called by the MISR Order Tool. These clients generate a unique identifier for each type of request that requires fault-handling provisions.

Many kinds of requests do not pose recovery issues and thus, do not employ request identifiers.

### **4.8.4 Request Responsibility**

The responsibility for the handling of recoverable requests by a server is given in the EED by determining if the request is synchronous or asynchronous.

#### **Synchronous Requests**

On a synchronous request, the application submitting the request is waiting for a response. Regardless of how the request is handled downstream, whether it succeeded or failed depends on the response the waiting application gets back. From its perspective, the request is not complete until it receives a response.

Therefore, if an EED application initializes a request and submits it synchronously, it has the responsibility for getting the request completed. This means if the request does not complete, for example, because the connection is lost to the server to which the request is submitted, the application needs to submit it again.

EED examples of synchronous interfaces include the Quick Server and Ftp Service.

#### **Asynchronous Requests**

When an application sends an asynchronous request, the receiving server is responsible for completing the request once it accepts the request. For example, the server may need to save the request (perhaps in a queue in a database) before sending an acknowledgment to the originating application. Of course, the server (Server A), can eventually complete

processing the request and pass it on to another server (Server B), also asynchronously. Once Server B accepts the request, it is responsible for seeing it to completion.

EED examples of asynchronous interfaces include the Order Manager Server and its component servers.

**Note:** Some servers, for instance the Data Access ESI Gateway Interface (EGI), are capable of handling both synchronous and asynchronous requests depending on the way the client decides to invoke the request.

#### 4.8.5 Queues

The reason queues are mentioned here is because they represent an important aspect of recovery. If a server uses queues to defer work until later, it needs to be concerned about what happens to the queue if the server crashes. The recovery rules in the request responsibility section state:

- If the server queues up synchronous requests, the client application is responsible for recovering the synchronous requests.
- If the server queues up asynchronous requests after accepting them, it is responsible for the asynchronous requests, which means, if a queue contains asynchronous requests, the server must make sure it can recover the queue in case of a crash.

Instead, a server handling asynchronous requests must keep a queue in a safe place so it can be recovered in case of a restart (such a restart that recovers the current requests is called a "warm start"). If a warm start takes place with asynchronous requests, the sending application does not even notice there was a problem. The processing gets completed eventually.

Note, however, that queued synchronous requests require special consideration: If a warm start takes place and some of the queued requests are synchronous, the sending application is generally aware of the failure (it had to rebind, See Client-server Rebinding Section). Since it did not receive a response, it re-submits the request. The server must recognize the request as a re-submission and either ignore it or - if it already completed by the time the re-submission is received - return the completion status as a response to this request. Moreover, servers that might handle a large number of concurrent synchronous requests have to be able to deal with a sudden spike of request submissions following a warm start, as their clients re-submit these requests.

A warm start can cause a problem; for instance, one of the active requests may be the reason the server crashed. This could result in a warm restart loop: each time the warm start is attempted the server crashes again because of the request. In such a case, operations can use a cold start to empty the queue of all requests (at the expense of having to recover queued asynchronous requests that were lost manually).

#### 4.8.6 Request Responses

Servers have the responsibility to classify a response appropriately. Client applications have the responsibility to process a response appropriately, depending on its type.

Client applications can pass the response on to the calling application (e.g., success, warning, or fatal error); or retry (retry error). At the beginning of a request chain, there may be a user or operator (if this is a user or operator submitted request). In this case, the error is passed back to the user/operator for action where possible.

Where this is not possible (e.g., system generated requests, or if a data order runs into an error after it was already accepted and the user/operator is no longer connected), errors are logged. They are brought to the attention of the DAAC operations staff for action if there is a corresponding server GUI for the operator. However, not all EED servers are associated with an operator GUI. In these cases, operators need to monitor the server logs for errors on a regular basis.

Failure events are classified as having any of three severity levels:

- Fatal errors,
- Retry errors and
- Warnings

Fatal errors are returned when a request cannot be serviced, even with operator intervention. For example, if a request is made to distribute data via FTP to a non-existent host, the request is failed with a fatal error.

Retry errors can be recovered from and such errors should be returned back to the client only when the server cannot recover from the error automatically. Retry errors may also necessitate operator assistance for recovery purposes, such as in the case of a tape left in a device that must be manually removed.

Warnings are provided where operations can proceed without interruption, but where an unexpected circumstance was detected. For example, if a client requests a file to be removed, and the file does not exist, there is no error, per se, but a warning is generated to caution the client the file to be removed did not exist in the first place.

The situation where a server does not return a response represents a special case. It can occur, for example, when an application calls a server and the server crashes before it can send a response or there is a communication error that prevents a response within a reasonable time. The situation is important because now the client application does not really know what happened to the request:

- a. Did it reach the server?
- b. Did the server start the request but not complete it?
- c. Did the server complete the request with an error but was not able to send the error response?
- d. Did the server process it successfully?

The EED recovery policy is that in such situations, the client application should either re-submit the request or if it is possible, return an appropriate error to the user/operator who submitted the request (to avoid leaving them with a hanging GUI while EED goes through endless retries).

Note that if the request did reach the server, the server now sees the request twice (i.e., this has become a duplicate request). Therefore, there need to be provisions to handle duplicate requests gracefully.

Table 4.8-1 summarizes the five categories of request responses, and the specific requirements for the application or server currently responsible for the request. EED servers have been directed to classify their responses accordingly.

**Table 4.8-1. Request Responses**

Request Response	Response Description
Success	The server sends back a message to acknowledge the successful completion of the request to the client. The request is considered complete.
Warning	This is provided where operations proceed without interruption, but where an unexpected circumstance is detected. The calling application needs to determine whether to alert the user or operator of the situation.
Error, retry the request	This can happen if the server encountered a temporary error condition, such as a media error on output. The request can be "retried" and the application responsible for the request should re-submit it after a suitable wait time. However, if the request does not succeed after a (configurable) number of retries, it should be considered "failed." If a GUI supports the application, the request may be suspended (if it makes sense to alert the operations staff to remedy the situation).
Error, cannot retry the request	This can happen if the server encounters an error condition that is sure to re-occur if the same request is submitted again. Examples might be a syntax error in the request (indicating some internal software problem), or an attempt to retrieve a non-existent granule. The request is considered "failed." The server responsible for the request sends back a failure notification. If a GUI supports the application, the request may be suspended (if it makes sense to get the operations staff involved at this point), but the operations staff may or may not be able to help.
No response returned by server	This can happen, for example, if the server to which the request was submitted crashes before a response or an acknowledgment is returned. In this case, the client can make no assumptions about the request. The client responsible for the request should send the request again or retry the request.

Transient errors such as network errors are always retry errors. In general, clients and servers that experience transient, retry errors can first attempt to recover by retrying the operation automatically. One special case of this is "rebinding". Rebinding refers to the process by which

a client automatically attempts to re-establish communications with a socket server in the event communications are disrupted. This disruption may be caused by transient network failure, or by the server being brought down or crashing. In any case, the client automatically attempts to reconnect to the server for a configurable period of time on a client-by-client basis.

EED processes that encounter an error or receive an error from a server request may either pass the error back to a higher-level client or present it to the operator for operator intervention. The fault handling policies are detailed in Table 4.8-2.

**Table 4.8-2. Fault Handling Policies (1 of 2)**

CI	Client Processes	Fault Handling Policy
DPLINGEST	EcDIInPollingService	<p>Retry errors: Errors are retried a configurable number of times for resources, then the resource is suspended. Examples of retrievable errors are connection failures and timeouts for file transfers.</p> <p>Fatal errors: Resources are suspended immediately for non-transient errors. Examples of non-transient errors are host address does not exist, and login to host failed.</p>
	EcDIInProcessingService	<p>Retry errors: Errors are retried a configurable number of times for resources, then the resource is suspended. Examples of retriable errors are connection failures and timeouts for quick server operations such as checksum and band extraction.</p> <p>Fatal errors: Resources are suspended immediately for non-transient errors. Examples of non-transient errors are quick server not running and failures to login to a transfer host.</p>
	EcDIInNotificationService	<p>Retry errors: Errors are retried a configurable number of times for resources, then the resource is suspended. Examples of retrievable errors are connection failures and timeouts for file transfers.</p> <p>Fatal errors: Resources are suspended immediately for non-transient errors. Examples of non-transient errors are host address does not exist, and login to host failed.</p>
SSS	EcNbSubscribedEventDriver EcNbActionDriver EcNbDeleteRequestDriver EcNbRecoverDriver	<p>All errors are logged.</p> <p>Failed attempts to connect to database are retried.</p> <p>Failed database queries are retried if the reason for failure was deadlock.</p>

**Table 4.8-2. Fault Handling Policies (2 of 2)**

CI	Client Processes	Fault Handling Policy
OMS	EcOmOrderManagerServer	<p>All errors are logged.</p> <p>Failed attempts to connect to database are retried.</p> <p>Retry errors: Errors are retried a configurable number of times, then passed back to the calling process.</p> <p>Fatal errors: Errors are logged and the request is suspended and operator intervention is generated. Operators then have a choice to hold, fail or resubmit the request.</p>
BMGT	EcBmBMGT	<p>All errors are logged in one of the BMGT log files. Errors for automatic cycles are always retried. Errors for all other types are retried unless they occur after generation has completed in which case they may be considered fatal. Database errors are retried to a configured retry limit, however serious database errors such as a missing stored procedure will cause a server to terminate. Repeated errors may suspend BMGT product generation and export and raise an alert in the operator GUI, depending on the type of error.</p>
DPL	EcDIActionDriver	<p>Retriable errors are retried a configurable number of times, after which the request will be failed. If the error is associated with a resource, such as the unavailability of a file system, the resource will be suspended, and all requests using that resource will also be suspended. The Action Driver will periodically test the associated resource, and automatically resume all affected requests once the resource becomes available again.</p> <p>Fatal errors, such as invalid granules, are failed immediately.</p> <p>All errors are logged in both application ALOG files, as well as in the database (a request that fails as a result of an error will have an error message recorded in the database).</p>
DPL	DataAccess	<p>Most of the Data Access services are purely synchronous and have no retry mechanism. Due to their 'on demand' nature, most types of errors will result in an error message being propagated all the way back to the initiating client. In the case of an asynchronous request to EGI, jobs within the request will not be retried, but a failure in a single job will not result in the entire request failing.</p>

### 4.8.7 Duplicate Request Detection

The above scheme for handling requests in cases of faults poses a potential problem. The request could have been re-submitted because there was no response returned by the server. But, in fact, the server completed the request but was unable to get the status back to the client (e.g., because of communications problems or a machine crash). The following measures are intended to deal with this situation:

- **Trivial duplicate requests.** There are many interfaces where sending a new request to retry a service whose outcome is unknown either has no or negligible impact on the EED. This is because many EED services have been designed with this goal in mind. In these cases, the new request will simply replace the old and have the same end result as the original request and no unintended consequences.
- **Recognize non-trivial duplicate requests.** Where executing the same request more than once can have undesirable consequences, EED provides a mechanism for recognizing re-submitted requests. Each request is tagged with a unique identifier (see Request Identification Section). Upon submission of a request, the receiving server of the request must check the identifier and recognize when it is a re-submission of a previous request it received. For example, the server may realize the request has been completed and simply acknowledges the successful completion.

### 4.8.8 Server Crash and Restart

- **Server Crash**

When a server crashes, the only impact on the system is that clients cannot continue to submit requests for processing. Synchronous requests in progress result in an exception being thrown back to the client process, which enters a rebinding failure recovery mode (see Client-Server Rebinding section above). Attempts to submit requests while the server is down results in the client blocking until a communications timeout has been reached.

- **Server Restart**

When a server restarts, it may perform various re-synchronization activities in order to recover from an unexpected termination. In the event of a server cold start or cold restart, the server also cancels all outstanding requests and reclaims all associated resources. Note that the distinction between cold start and cold restart is described in the section above on Start Temperature. Specifics of server startup behavior are detailed in Table 4.8-3. Unless otherwise stated, existing request queues are always retained for warm restarts and cleared for cold starts or cold restarts.

**Table 4.8-3. Server Response versus Restart Temperature**

CI	Server(s)	Special Behavior on Warm Restart	Special Behavior on Cold Start or Cold Restart
DPLINGEST	EcDIInPollingService	None.	None.
DPLINGEST	EcDIInProcessingService	All ingest requests that did not reach a terminal state in the previous processing run will be re-queued in processing and executed from their last persisted state.	All ingest requests that did not reach a terminal state in the previous processing run will be moved to the state 'TERMINATED'. They will not be re-queued.
DPLINGEST	EcDIInNotificationService	None.	None.
SSS	EcNbSubscribedEventDriver EcNbActionDriver EcNbDeleteRequestDriver EcNbRecoverDriver	N/A	N/A
OMS	EcOmOrderManagerServer	Full Recovery	N/A
DPL	EcDIActionDriver	The Action Driver recovers all existing requests from the database.	N/A The Action Driver does not have a cold restart capability.
DPL	ESI Gateway Interface	The EGI recovers all existing requests from the database	N/A

- **Request Re-submission**

Upon restarting a crashed client or server, requests are typically re-submitted. If the restarted process was started warm, the fault recovery capabilities permit the server to resume processing of the request from its last check-pointed state. This prevents needless repetition of potentially time-consuming activities. Specific behavior of servers upon re-submission of a request is detailed in Table 4.8-4. Note that a cell value of N/A means the server either has no clients or the clients do not re-submit requests.

**Table 4.8-4. Server Response for Request Re-submission**

CI	Server(s)	Behavior on Request Re-submission
DPLINGEST	EcDIInPollingService EcDIInNotificationService	N/A
DPLINGEST	EcDIInProcessingService.	The newly resubmitted request will have the same requestid and continue being processed from the last check-pointed state from the last processing run.
SSS	EcNbEventDriver EcNbActionDriver EcNbDeleteRequestDriver EcNbRecoverDriver	There is no resubmission of requests. EcNbRecoverDriver monitors the SSS database for events or actions that did not run to completion and re-enqueues them.
OMS	EcOmOrderManagerServer	Incomplete requests in OMS are picked up and processed upon restarting OMS Server. The incomplete requests have the same requestid
DPL	EcDIAActionDriver	N/A The Action Driver is asynchronous, with requests submitted via the database. Clients cannot detect whether or not the Action Driver is executing. However, normal operations frequently results in duplicate requests (requests for the same granule) from different clients, and the Action Driver is therefore designed to handle this.
DPL	ESI Gateway Interface	Resubmitted requests will be processed as new requests.

#### 4.8.9 Client Crash and Restart

- **Client Crash**

When a client crashes in the EED system, fault recovery-enabled servers have several possible responses. Servers may continue to service client requests, independent of the client's status. Servers may choose to suspend processing of client requests, but permit the requests to be resumed upon client recovery. Or, servers may terminate servicing of the client requests, canceling all work done on the requests. The behavior of each CI is detailed in Table 4.8-5. Note that the behavior of a server in the event of a client crash does not vary from client to client.

**Table 4.8-5. Server Responses to Client Failures**

CI	Server(s)	Behavior on Client Crash
DPLINGEST	EcDIInProcessingService EcDIInNotificationService	Requests in process are serviced to completion.
	EcDIInPollingService	N/A
SSS	EcNbSubscribedEventDriver EcNbActionDriver EcNbDeleteRequestDriver EcNbRecoverDriver	Processing is database driven and not influenced by outside processes.
OMS	EcOmOrderManagerServer	Requests in process are serviced to completion.
DPL	EcDIActionDriver	Requests in process are serviced to completion.
DPL	ESI Gateway Interface	Requests in process are serviced to completion

- **Client Restart**

Client restart behavior depends upon the type of client. For command line tools used by operators, it is generally the responsibility of the operator to decide whether or not the client needs to be restarted. Server processes acting as clients to other servers or processes will generally determine how to restart processing of the request using persisted checkpoints (for example, in the database).

**Some known limitations within the EED are:**

- a.) Requests with many sub-requests can experience timing problems because of nested retries or because one of the requests is suspended.
- b.) Coding errors can cause unanticipated fault behavior that is different from what is described above (and such occurrences should be reported as NCRs.)
- c.) System engineers and designers may have made mistakes in classifying errors (e.g., as fatal versus retry.)
- d.) Not all EED applications use the error recovery capabilities of the EED database interface infrastructure.

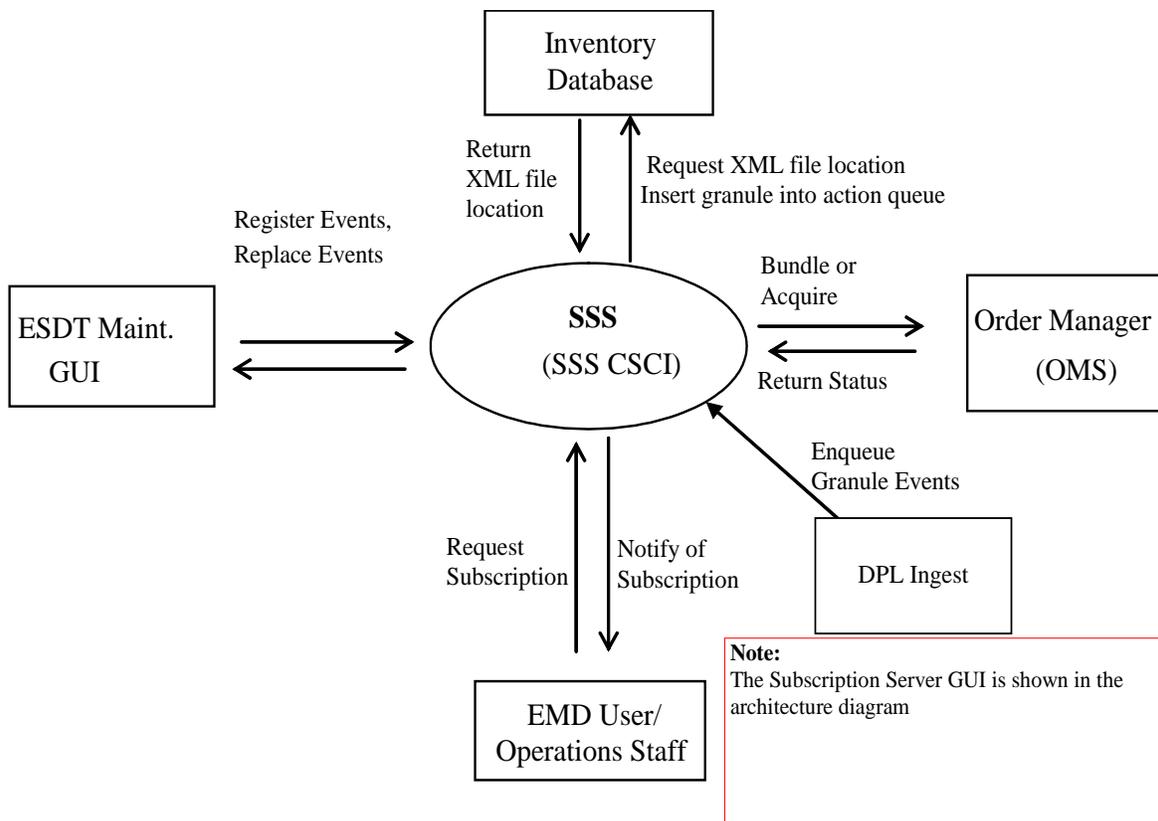
## 4.9 Spatial Subscription Server (SSS) Subsystem Overview

The Spatial Subscription Server (SSS) subsystem is the principal means by which users can establish standing orders for data. Users enter subscriptions for specific ESDTs using a GUI or command line interface (CLI). A subscription may be qualified by specifying one or more constraints on the metadata of matching granules. This includes the capability of qualifying the subscription spatially by specifying a geographic area (rectangle) over which the data was collected. A subscription has one or more associated actions such as data distribution, email notification, Data Pool publication, or bundling, i.e. adding a granule to an Order Manager bundle.

In addition to the subscription creation components, the SSS subsystem is comprised of a schema installed in the “ecs” database, and four runtime drivers: an event driver to match subscriptions with granule events, an action driver to execute the actions of matched subscriptions, a recovery driver to restart stalled events or actions, and a deletion driver to clean up the database.

### Spatial Subscription Server (SSS) Context

Figure 4.9-1 is the Spatial Subscription Server context diagram. Table 4.9-1 provides descriptions of the interface events in the Spatial Subscription Server context diagram.



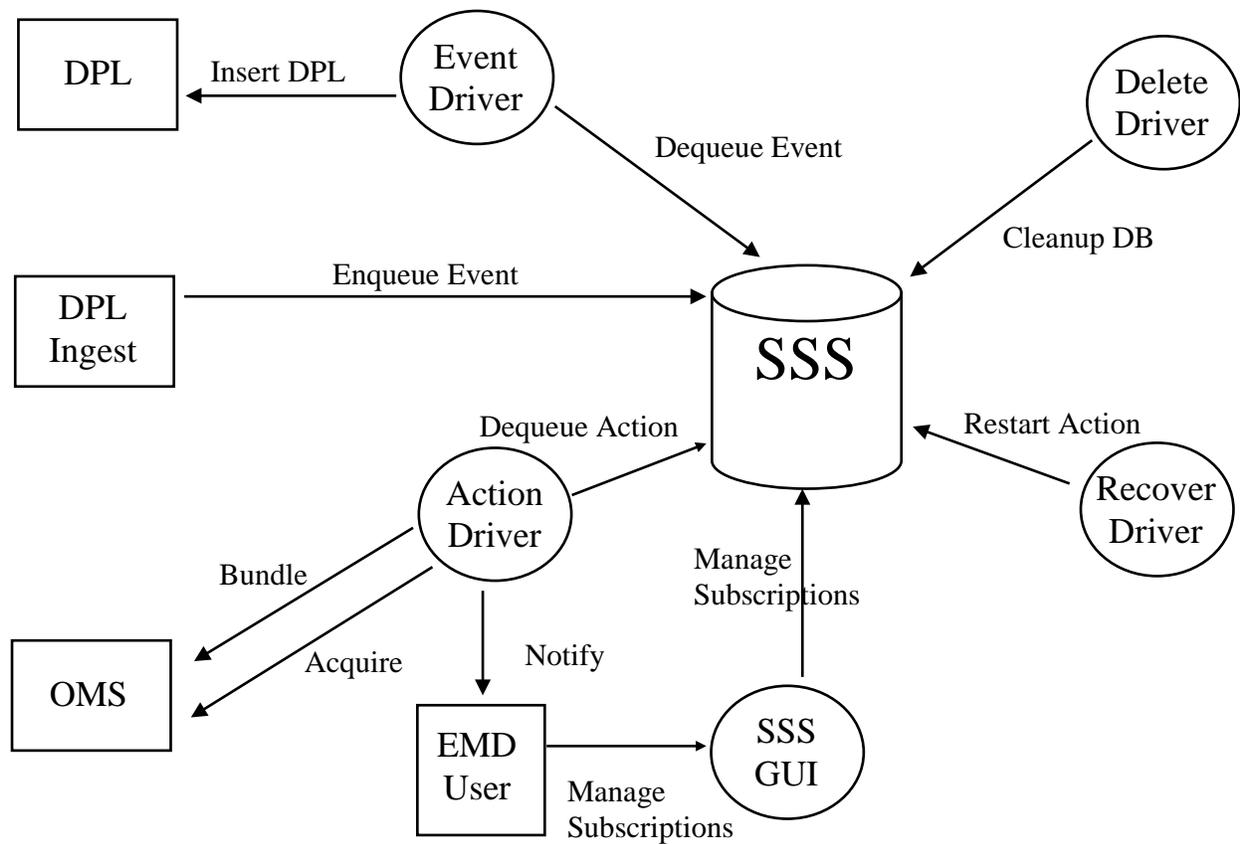
**Figure 4.9-1. Spatial Subscription Server Context Diagram**

**Table 4.9-1. Subscription Server Interface Events**

Event	Interface Event Description
Request XML file location	The <b>Inventory database</b> contains information about new granules, including the location of their XML metadata files. The SSS event driver queries the Inventory database for this information.
Notify of Subscription	The SSS CSC sends email notification to the <b>EMD User</b> when the subscribed event occurs, provided that a notification action was requested in the subscription.
Request Subscription	A subscriber ( <b>EMD user</b> requests <b>Operations Staff</b> to create the subscription) sends information (ESDT and, optionally, acceptable metadata values) with the subscription, specifying one or more actions (e.g., acquire and/or notification) to be taken when the subscribed event occurs.
Return status	Status returned by a stored procedure to indicate whether or not the call succeeded.
Register Events	The <b>ESDT Maintenance GUI</b> inserts information about an Earth Science Data Type (ESDT) into the SSS database when an ESDT is installed into the system.
Enqueue Granule Events	DPL Ingest will enqueue new granule events via an SSS stored procedure call.
Replace Events	The <b>ESDT Maintenance GUI</b> modifies the SSS database when an ESDT is deleted from the system.
Bundle or Acquire	SSS notifies OMS, via a stored procedure call, when a granule has matched a subscription. If the subscription is bundled, i.e. associated with an OMS bundling order, then the granule is inserted into the appropriate OMS bundle. If the subscription is not bundled, then an acquire request is sent to OMS.
Insert granule into action queue	If a subscription has an associated Data Pool action, then SSS will insert a row into the Data Pool action queue table, indicating that the granule that matched the subscription should be published in the Data Pool.
Send theme validation	If a subscription's Data Pool action is associated with a Data Pool theme, then the Data Pool will verify that the theme exists and is enabled for insert.

#### 4.9.1 Spatial Subscription Server Architecture

Figure 4.9-2 is the Spatial Subscription Server architecture diagram. The diagram shows the events sent to the Spatial Subscription Server processes and the events the Subscription Server processes send to other processes.



**Figure 4.9-2. Spatial Subscription Server Architecture Diagram**

Table 4.9-2 provides descriptions of the processes shown in the Spatial Subscription Server architecture diagram.

**Table 4.9-2. Spatial Subscription Server Processes**

Process	Type	Hardware CI	COTS/ Developed	Functionality
EcNbSubscribedEventDriver ("Event Driver")	Server	OMSHW	Developed	The SSS event driver dequeues events and matches them with active subscriptions. Information about matched subscriptions is placed in the action queue. If a matched subscription has a Data Pool action, the event driver inserts information into the Data Pool database.
EcNbActionDriver ("Action Driver")	Server	OMSHW	Developed	The SSS action driver dequeues matched subscriptions and executes their associated actions (acquire or notification). An acquire is directed to the Order Manager. If a subscription is bundled, then the granule that matched it is added to that bundle via an OMS interface.
EcNbDeleteRequestDriver ("Delete Driver")	Server	OMSHW	Developed	The SSS delete driver dequeues from the delete request queue and cleans up database storage for the completed action or event.
EcNbRecoverDriver ("Recover Driver")	Server	OMSHW	Developed	The SSS recover driver monitors the event and action queues for stalled events/actions and reenqueues them so that they will be tried again.
EcNbSubscriptionGUI ("SSS GUI")	GUI	OMSHW	Developed	The SSS GUI provides an operator interface for submitting, updating and deleting subscriptions. It is also used for creating OMS bundling orders and for bundling subscriptions to bundling orders.

EMD Baseline Information System (EBIS) Document 920-TDx-001 (Hardware Design Diagram) provides descriptions of the HWCI, and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

#### **4.9.1.1 Subscription Server Process Interface Descriptions**

Table 4.9-3 provides descriptions of the interface events shown in the Subscription Server architecture diagram.

**Table 4.9-3. Spatial Subscription Server Process Interface Events (1 of 2)**

Event	Event Frequency	Interface	Initiated by	Event Description
Enqueue Event	Once per granule ingest	<i>Process:</i> ProcSubscribedEventEnqueue	<i>Process:</i> DPL Ingest	The stored procedure is called when a new granule is ingested by DPL Ingest.
Dequeue Event	Once per event	<i>Process:</i> ProcSubscribedEventDequeue	<i>Process:</i> EcNbSubscribedEventDriver	An event driver instance will dequeue up to 10 events from the event queue at one time. It will then process the events sequentially by getting the metadata for each granule and comparing it with the list of active subscriptions. If a subscription matches a granule event, information about the match is placed into the action queue.
Insert DPL	Once per event	<i>Process:</i> TrigInsEcNbDpEventDetails	<i>Process:</i> EcNbSubscribedEventDriver	When a granule event matches one or more subscriptions, at least one of which has an associated Data Pool action, the event driver will insert information about the granule (with subscription numbers) into the Data Pool database. A single insert per event is performed by an insert trigger on the table EcNbDpEventDetails.
Dequeue Action	Once per matched subscription	<i>Process:</i> ProcActionDequeue	<i>Process:</i> EcNbActionDriver	An action driver instance will dequeue up to 10 matched subscriptions from the action queue at one time. It will then process them sequentially by getting the actions for each subscription. If the subscription is bundled, then the granule is added to the current bundle for that bundling order via a stored procedure call to the OMS database. Otherwise, the action driver will initiate an acquire of the granule or send email notification to the user, depending on how the subscription was set up.

**Table 4.9-3. Spatial Subscription Server Process Interface Events (2 of 2)**

Event	Event Frequency	Interface	Initiated by	Event Description
Acquire	Once per matched subscription	<i>Process:</i> OmCreateNonBundlingOrder (OMS case)	<i>Process:</i> EcNbActionDriver	If a matched subscription has an associated acquire action, the action driver will initiate the acquire by a stored procedure call to OMS.
Bundle	Once per matched subscription	<i>Process:</i> OmInsertBundleRequest	<i>Process:</i> EcNbActionDriver	If a matched subscription is a bundled subscription, the action driver will send information about the granule to OMS via a stored procedure call.
Notify	Once per matched subscription	<i>Process:</i> mailx	<i>Process:</i> EcNbActionDriver	If a matched subscription has an associated notification action, the action driver will compose an email message and send it to the address specified in the subscription definition.
Restart Action	Once per action or event	<i>Process:</i> ProcActionReEnqueue, ProcSubscribedEventReEnqueue	<i>Process:</i> EcNbRecoverDriver	If an action or event appears to have stalled, i.e. did not run to completion based on evidence in the log tables, the recover driver will reenqueue the action or event in its appropriate queue.
Cleanup DB	Once per action or event	<i>Process:</i> ProcDequeueDeleteRequest, ProcDeleteProcessedSub, ProcDeleteProcessedEvent	<i>Process:</i> EcNbDeleteRequestDriver	The delete driver will clean up tables in the database based on entries in the delete request queue. Each entry in this queue corresponds to one action or one event.
Manage Subscriptions	Various	<i>Process:</i> EcNbSubscriptionGUI	<i>Process:</i> EcNbSubscriptionGUI	The SSS GUI allows a user to create, delete, edit or view subscriptions. Or to create, delete, edit or view bundling orders and bundle subscriptions to them.

#### 4.9.1.2 Subscription Server Data Stores

Spatial Subscription Server uses a COTS software RDBMS for the storage of persistent data. The following is a brief description of the principal types of data contained in the database:

- **Attributes:** includes the ESDTs for which subscriptions can be created and the metadata attributes that can be used to qualify those subscriptions.
- **Subscriptions:** information about subscriptions that have been created for users, their associated qualifying expressions, and their associated actions.

- **Events:** information about newly arrived data granules, their metadata, and the subscriptions that match them.
- **Actions:** information about actions for matched subscriptions that need to be carried out, e.g. acquire or email notification.

## 4.10 Data Pool Subsystem Overview

The Data Pool is a large online archive of ECS data at each DAAC. Science, metadata (in xml format), and browse files (in jpg format) are stored in the public Data Pool.

Hidden directories in the Data Pool file systems (/datapool/<mode>/user/<fs>/orderdata) are used as staging areas for all granules being inserted into the Data Pool, granules whose collections are configured hidden and for browse and processing output distributed via the Order Management Subsystem (OMS).

The Data Pool subsystem consists of the following components and supporting utilities:

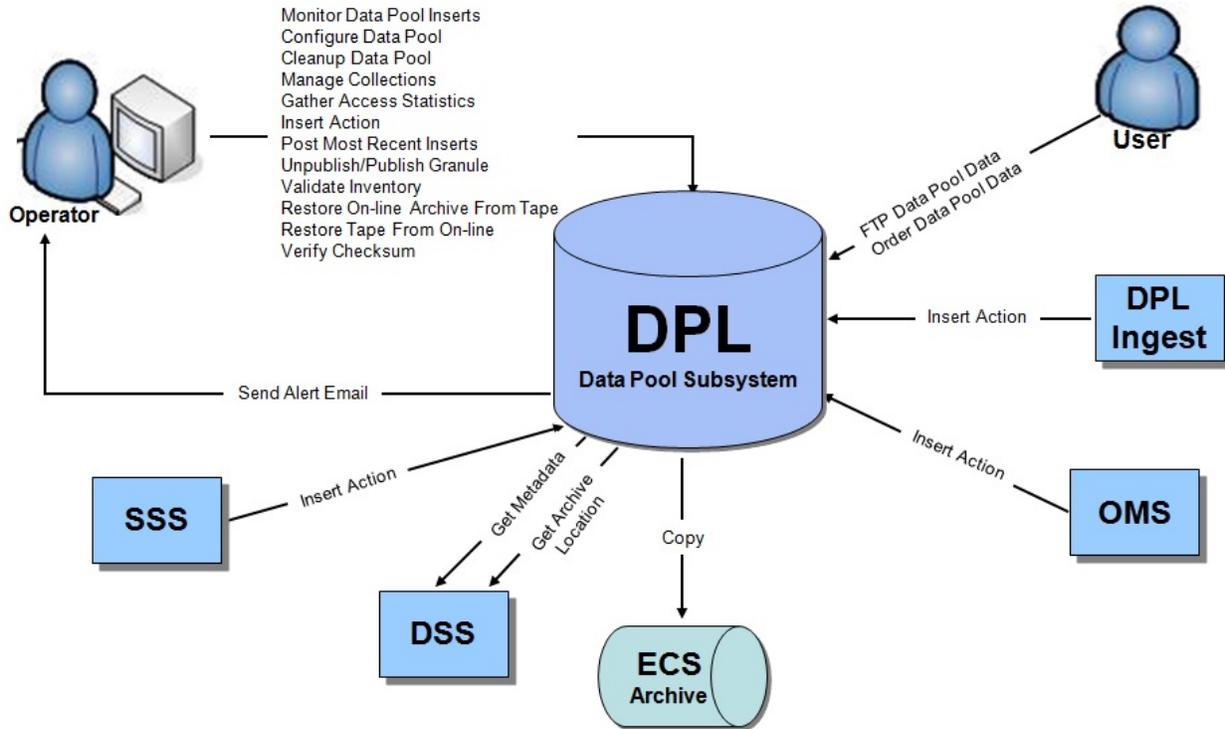
1. **Data Pool Insert:** inserts ECS data into the Data Pool. ECS data is copied from the ECS archive into the Data Pool, based on an ECS granule id. The Inventory database inventory is updated for each granule inserted in the Data Pool. Data Pool Insert consists of six major subcomponents:
  - a) The Data Pool Action Driver (DPAD): a C++ executable which schedules Data Pool insert actions based on a queue of Data Pool insert actions populated by the Spatial Subscription Server, the Data Pool Publish Utility, Data Pool Ingest, the Order Manager Server, or the Migration processes.
  - b) The Data Pool Insert Utility (NDPIU), a java executable which manages the registration and publication of an ECS data granule into the Data Pool;
  - c) The Data Pool Quick Server, a C++ executable which is installed on the ECS service hosts. The Quick Server is used by the DPAD to perform copy and checksum operations. It is also used by DPL Ingest and OMS to perform operations which are performed on ECS service hosts for load balancing reasons, or which cannot be performed on the local host due to lack of data access (mount points, etc.)
  - d) The band extraction utility (bandtool), a C executable invoked by the DPAD, which extracts band information from HDF-EOS granules and stores the extracted information in a .BandHeader file in the temp area on the ESDT file system. The .BandHeader file is used by the NDPIU during granule registration. The bandtool is invoked only if the granule being inserted is from a collection eligible for conversion by the HDF-EOS to GeoTiff Conversion Tool (HEG);
  - e) The jpeg extraction utility (hdf2jpeg), a C executable invoked by the NDPIU, which extracts browse images (jpeg or raster) from a browse hdfEOS file on browse publication.
2. **Data Pool On-line Archive Cleanup and Validation:** As of 8.1, the cleanup of ECS granules is handled by the Inventory Deletion suite of utilities. The validation of ECS Inventory has been grouped and built into separate scripts for performance reasons. Additional validation against the Inventory database is added as part of the on-line archive capability.
  - a) **EcDICleanupFilesOnDisk.pl:** a perl utility, which reports and fixes inconsistencies between Data Pool directories and the database, namely orphan/phantom validation and orphan cleanup.

- b) **EcDICleanupGranules.pl**: a perl utility, which cleans non-ECS granules from Data Pool on-line archive and the database. It also supports “error recovery” scenarios to remove granules that failed publishing and were left in an unsupported state.
  - c) **EcDIInventoryValidationTool.pl**: a perl utility, which identifies discrepancies in the AIM Inventory Database.
  - d) **EcDIlinkCheck.ksh**: a korn shell script, which provides report on invalid soft links in Data Pool on-line archive and optionally remove them.
  - e) **EcDIXcu.pl**: a perl utility, which checks for corruption of XML files in the Data Pool.
3. **Data Pool Maintenance GUI (EcDIDpm)**: a perl-based web GUI that allows DAAC operations staff to monitor Data Pool insert activity and to control the Data Pool configuration.
  4. **Data Pool Access Statistics utilities**: Rollup scripts - perl utilities which parse firewall ftp and http logs (EcDIRollupWuFtpLogs.pl and EcDIRollupHttpLogs.pl) for accesses to the Data Pool directories, and then roll up access information for storage in the Inventory database. Maintenance Scripts – perl utilities which are used for archiving, deleting, and backing up granule access data in the Inventory database.
  5. **Data Pool FTP Server**: customized wu-ftp daemon, which supports ftp access to Data Pool directories and also provides a checksum-on-download service.
  6. **Data Pool Update Granule Expiration utility (EcDIUpdateGranule.pl)**: a perl utility, which allows operations staff to update the Data Pool expiration date and retention priority for specified Data Pool non-ECS granules.
  7. **Data Pool Most Recent Insert Utility (EcDIMostRecentInsert.pl)**: a perl utility, which creates files at the file system and data collection level of the Data Pool directory structure which contain information about granules recently inserted into the Data Pool at those levels.
  8. **Data Pool Collection Remapping Utility (EcDIRemap.pl)**: a perl utility, which allows DAAC operations staff to remap all data in a Data Pool collection directory from one higher level collection group directory to another.
  9. **Data Pool Move Collection Utility (EcDIMoveCollection.pl)**: a perl utility, which allows DAAC operations staff to move a Data Pool collection from one file system to another.
  10. **Data Pool Hidden Scrambler Utility (EcDIHiddenScrambler.pl)**: a perl utility, which creates new names for specified hidden directories, saves these names, renames the existing hidden directories, and updates existing FTP Pull links that point to the previous hidden directories to point to the corresponding renamed directory.
  11. **Data Pool Checksum Verification Utility**: A java-based stand-alone utility which can verify the integrity of files in the Data Pool using checksums. The utility could be set up as a background process as well as run on-demand by the DAAC operator to verify checksum values for a particular set of files.

12. **Data Pool Checksum Verification Server:** A C++ based stand-alone utility which can verify the integrity of files in the Data Pool using checksums. It provides resource management and load balancing to achieve optimal checksum throughput when the system is under load and normal operation. It replaces “Data Pool Checksum Verification Utility”.
13. **Data Pool Restore On-line Archive From Tape Utility:** A java-based stand-alone utility which performs bulk repairs of the on-line archive, especially in the case of serious disk errors or a loss of a Data Pool file system. It can also be used to restore the integrity of granules which have files missing or corrupted, or missing links.
14. **Data Pool Restore AIM Tape Archive From On-line Archive Utility:** A java-based stand-alone utility which provides bulk repair as well as individual science granules in the AIM tape archive by replacing science granules with their copy from Data Pool On-line archive.
15. **Data Pool Publish Utility:** A java-based utility which allows operations staff to submit ECS insert actions for publication or registration, optionally with load control. The utility has been enhanced to include theme option and non-ECS insert. It includes all the capability from “Batch Insert Utility” (deprecated) and has more of its own.
16. **Data Pool Unpublish Utility:** A java-based utility which moves granules from the public Data Pool on-line archive to the hidden Data Pool on-line archive.
17. **Data Pool Ftp Service (EcDIftpService):** A C++ based stand-alone SOAP web service running on the ECS service hosts. It accepts Ftp service requests, processes the requests and sends the results back to the web service client. This service has ability to keep the FTP connections in the pool and reuse them later. It is used by DPL Ingest, OMS and BMGT to perform Ftp Transfer operations.

#### 4.10.1 Data Pool Subsystem Context

Figure 4.10-1 is the Data Pool Subsystem context diagram. The diagram shows the interaction of the Data Pool Subsystem with other EED subsystems. Table 4.10-1 provides descriptions of the interface events shown in the Data Pool Subsystem context diagram.



**Figure 4.10-1. Data Pool Subsystem Context Diagram**

**Table 4.10-1. Data Pool Subsystem Interface Events (1 of 3)**

Interface Event	Interface Event Description
Send Alert Email	The Data Pool Action Driver sends an alert email to a configured email address to notify operators of problems with an ECS Service Host, an archive file system, or a Data Pool file system.
Monitor Data Pool Inserts	The operator uses the Data Pool Maintenance GUI to monitor the queue of Data Pool inserts and to monitor the active insert processes.
Configure Data Pool	The operator uses the Data Pool Maintenance GUI to set values of Data Pool configuration parameters, and to define Data Pool entities such as themes and compression algorithms.
Cleanup Data Pool	The operator runs the Data Pool Cleanup utilities to clean specified granules out of the Data Pool on-line archive, and to identify and cleanup granules, which are orphans (on Data Pool disk but not in the database) or phantoms (in the database but not on disk), or invalid links(soft links point to invalid files).
Manage Collections	The operator uses the Data Pool Maintenance GUI to add, remove, or change specifications for Data Pool collections. The operator uses the Remap Collection utility to map a collection from one collection group to another. The operator uses the Move Collection utility to move a collection from one file system to another.

**Table 4.10-1. Data Pool Subsystem Interface Events (2 of 3)**

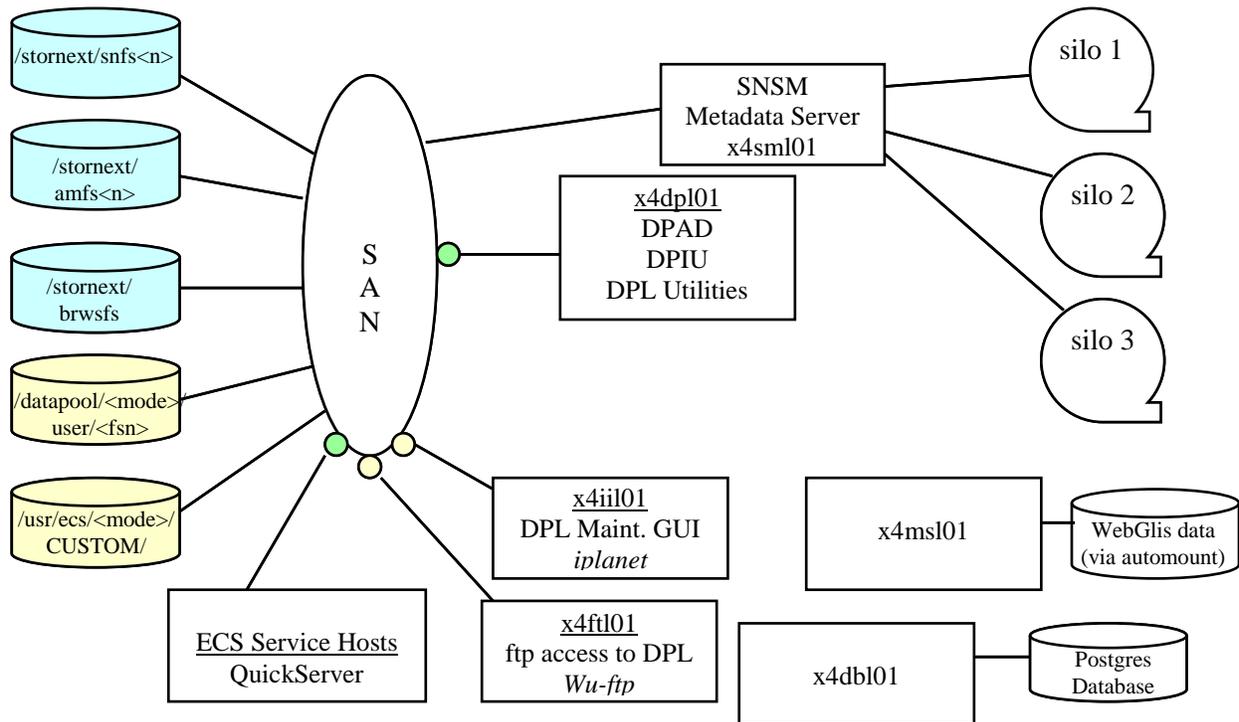
Interface Event	Interface Event Description
Gather Access Statistics	The operator uses the access statistics rollup scripts for the firewall ftp and http access logs to gather statistics about end user access to data pool files, and to store those statistics in the Inventory database.
Insert Action	The operator uses the Data Pool Publish utility to insert historical data from the ECS archive into the Data Pool.
Update Granule Expiration	The operator uses the Update Granule Expiration utility to update the expiration date or retention priority for one or a set of non-ECS granules.
Post Most Recent Inserts	The operator uses the Most Recent Inserts utility to post information about recent Data Pool Inserts to the Data Pool ftp directories.
Unpublish Granule	The operator uses the Data Pool Unpublish Utility to move granules from public Data Pool on-line archive to hidden Data Pool on-line archive.
Validate Inventory	The operator uses the Data Pool Inventory Validation Utility to identify discrepancies in the AIM Inventory Database.
Restore On-line Archive From Tape	The operator uses the Restore On-line Archive From Tape Utility to restore the integrity of granules to on-line archive from tape.
Restore Tape From On-line Archive	The operator uses the Restore Tape From On-line Archive Utility to restore the integrity of granules to tape from on-line archive.
Verify Checksum	The operator uses the Data Pool Checksum Verification Server to verify the integrity of the granules in Data Pool on-line archive.
FTP Data Pool data	The end user uses the customized WU-FTP service to download Data Pool data.
Order Data Pool data	The end user orders Data Pool data for ftp and http distribution. The end user may choose to convert, reformat, or subset the data using the HDF-EOS to GeoTiff Conversion Tool (HEG).
Insert Action	The Data Pool Ingest subsystem inserts a Data Pool insert action into the Data Pool Insert Action Queue (DIInsertActionQueue) for granules which are configured to be published in the Data Pool.
Insert Action	The OMS subsystem inserts a Data Pool insert action into the Data Pool Insert Action Queue (DIInsertActionQueue) for granules to be staged to the Data Pool for ECS distribution requests.
Copy	The DPL subsystem copies data from the ECS Archive to the appropriate Data Pool file system.
Get Archive Location	The DPL subsystem looks up archive location information in the Inventory database, for granules which will be copied from the ECS archive to the Data Pool.
Get Metadata	<p>The DPL subsystem gets metadata about ECS granules (QA, PH, etc.,) from the Inventory database, and uses this metadata to store corresponding metadata in the Inventory database and to create an xml metadata file on Data Pool disk.</p> <p>The DPL subsystem gets metadata path about ECS granules (SCIENCE) from Inventory database, and uses this path to get xml files from small archive to DataPool filesystem.</p>

**Table 4.10-1. Data Pool Subsystem Interface Events (3 of 3)**

Interface Event	Interface Event Description
Insert Action	The Spatial Subscription Server subsystem inserts Data Pool insert actions in the Data Pool Insert Action Queue (DIInsertActionQueue) for granules which are being inserted into the ECS inventory for which a Data Pool insert subscription is placed. Data Pool insert subscriptions are qualified subscriptions (unqualified Data Pool insert subscriptions have been replaced by DPL Ingest configuration of ESDTs for public Data Pool insert.)

### 4.10.2 Data Pool Hardware Context

Figure 4.10-2 is the Data Pool hardware context diagram. The diagram shows the interaction of the Data Pool custom code and COTS (in *italics*) with EED hardware components.



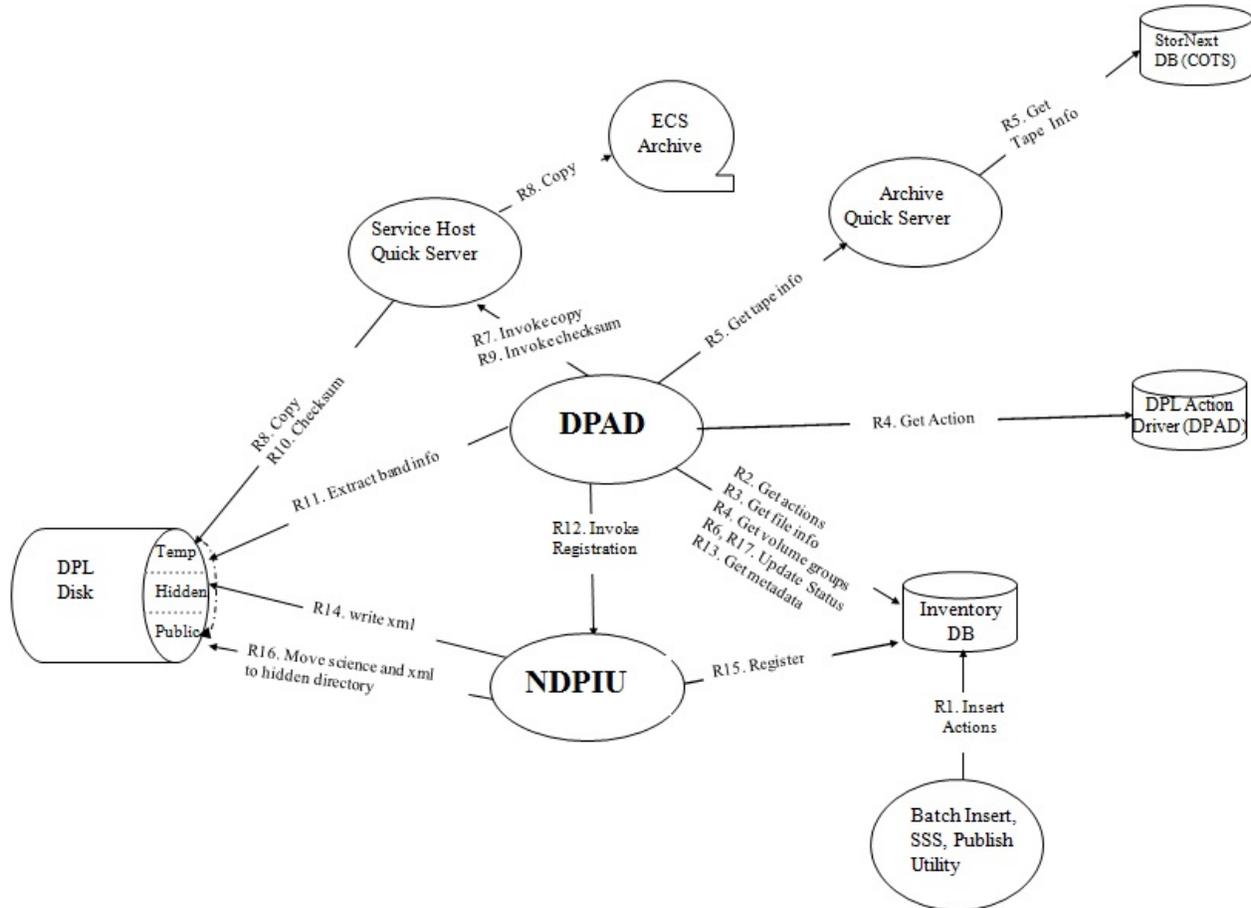
**Figure 4.10-2. Data Pool Hardware Context**

### 4.10.3 Data Pool Insert CSCI Functional Overview

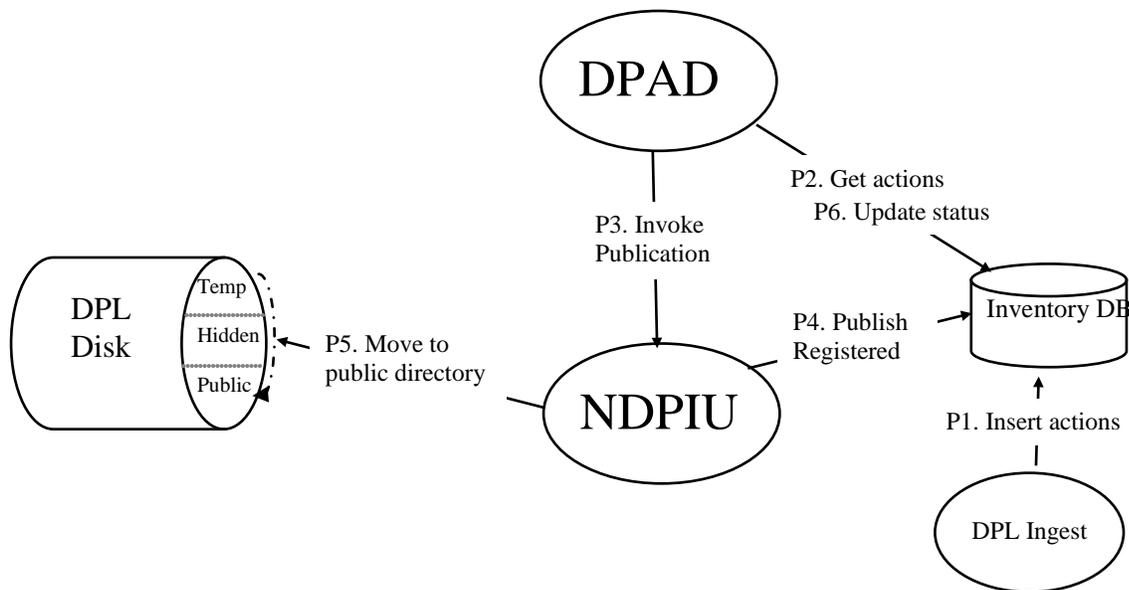
ECS granules are inserted into the Data Pool via a two-step process. The first step, registration, involves storing basic inventory information about the granule, needed by EED custom code applications, in the AIM Inventory Database, and copying the granule to a “hidden” directory

structure (datapool/<mode>/user/<filesystem>/orderdata) in the Data Pool. The second step, publication, occurs only for granules which belong to collections configured to be placed in the public Data Pool, where they are available for anonymous ftp download. During the publication step, the granule is moved from the Data Pool hidden directory structure to the public directory structure, where it can be accessed via anonymous ftp.

A functional overview of the two-step Data Pool Insert process for ECS granules is shown below in two diagrams. The first diagram (Figure 4.10-3) shows the process for registration of a granule in the Data Pool. The second diagram (Figure 4.10-4) shows the process for publication of a granule in the Data Pool.



**Figure 4.10-3. Data Pool Insert CSCI Architecture Diagram - Registration**



**Figure 4.10-4. Data Pool Insert CSCI Architecture Diagram - Publication**

There are five use cases for the Data Pool Insert process, one for each ECS process or component which requests insertion of a granule into the Data Pool. These use cases, and their relationship to the registration and publication steps, is shown in Table 4.10-2.

**Table 4.10-2. Use Cases for Data Pool Insert (1 of 2)**

Requestor	Context	Data Pool Insert processes
OMS	Distributes granules in the Data Pool.	Registration (events R1 – R17 in Table 4.10-3 for data type such as QA, PH, DAP, etc; events R1-R12 and R15-R17 in Table 4.10-3 for science granules).
Data Pool Ingest	requests publication of a granule in the Data Pool after the granule has been ingested and archived	Publication (events P1 – P6 in Table 4.10-4).
Publish Utility	Queues existing ECS and non-ECS granules for publication or registration in Data Pool on-line archive	Registration Publication (events R1-R16 in Table 4.10-3 for data type such as QA, PH, DAP, etc; events R1-R12 and R15-R16 in Table 4.10-3 for science granules, and P3-P6 in Table 4.10-4)

**Table 4.10-2. Use Cases for Data Pool Insert (2 of 2)**

Requestor	Context	Data Pool Insert processes
Spatial Subscription Server	queues granules for insertion into the public Data Pool as a result of a qualified subscription with a Data Pool insert action	Registration Publication (events R1-R16 in Table 4.10-3 for data type such as QA, PH, DAP, etc; events R1-R12 and R15-R17 in Table 4.10-4 for science granules, and P3-P6 in Table 4.10-4)

Note that Data Pool Ingest also stages granules in the hidden Data Pool during granule ingest. That process is somewhat different than the Registration process described below, in that it uses a different invocation of the NDPIU and does not involve the DPAD. Data Pool Ingest staging of granules in the hidden Data Pool is documented in the Data Pool Ingest chapter of this document.

Table 4.10-3 provides a process description for each of the major custom code components of the Data Pool insert process. Table 4.10-4 describes the interface events among the Data Pool insert process components for registration and publication.

**Table 4.10-3. Data Pool Insert CSCI Process Description (1 of 2)**

Process	Type	Hardware CI	COTS/ Developed	Functionality
EcDIActionDriver	Server	DPLHW	Developed	EcDIActionDriver (DPAD) is a C++ server that is responsible for dispatching Data Pool insert actions for ECS granules from the insert action queue in the AIM Inventory database (DIInsertActionQueue) as well as performing registration or publication of ECS granules (possibly involving copy, checksum, band extraction operations).
EcDIInsertUtility	Java utility	DPLHW	Developed	EcDIInsertUtility (NDPIU) is a java executable that is invoked by the EcDIActionDriver to register and publish ECS granules in the Data Pool. It populates AIM database, and it moves files to hidden or public Data Pool. One copy of the EcDIInsertUtility is invoked for each ECS granule to be inserted in the Data Pool.

**Table 4.10-3. Data Pool Insert CSCI Process Description (2 of 2)**

Process	Type	Hardware CI	COTS/ Developed	Functionality
EcDIQuickServer	Server	ACMHW, DIPHW, DRPHW, SPRHW, MSSHW, AITHW, CSSHW, DMGHW, DPSHW, INTHW, DPLHW, OMSHW	Developed	The EcDIQuickServer (Service Host Quick Server) is a C++ server which performs CPU-intensive operations, such as copy and checksum, on ECS service hosts.
hdf2jpeg	Utility	DPLHW	Developed	Java utility that extracts jpgs from an HDFEOS granule.
bandtool	Utility	DPLHW	Developed	C utility that extracts band information from an HDFEOS granule.

**Table 4.10-4. Data Pool ECS Insert CSCI Process Interface Events (1 of 4)**

Event	Event Frequency	Interface	Initiated By	Event Description
R1. Insert Action	One per granule inserted in Inventory which qualifies for existing subscription with Data Pool Insert action	Database: Inventory (DIInsertActionQueue)	Trigger: TrigInsEcNbDpEventDetails. sql	When a granule is inserted into Inventory which matches an existing subscription with Data Pool Insert action, the trigger inserts a row into the DIInsertActionQueue in the AIM Inventory database with actionSource = null.
R1. Insert Action	One per granule in Syn IV order placed through Order Manager	Database: Inventory	Process: OmServer Stored Proc: OmInsDPLAction	When a granule is ordered in Syn IV mode via the Order Manager, OMS inserts a row into the DIInsertActionQueue in the AIM Inventory database, with actionSource = O.
R1. Insert Action	One per granule in input file for the Publish utility.	Database: Inventory	Utility: EcDIPublishUtility	For each valid granule in its input file, the Batch Insert Utility inserts a row into the DIInsertActionQueue in the AIM Inventory database, with actionSource = B or R.
R1. Insert Action	One per granule in input file, or on the command line for the Publish utility.	Database: Inventory	Utility: EcDIPublishUtility	For each valid granule in its input file or on the command line, the Publish Utility inserts a row into the DIInsertActionQueue in the AIM Inventory database, with actionSource = B or R.
R2. Get Action	Continuously, as long as there are actions in DIInsertActionQueue with status = null or status = RETRY. If no actions, once per configured time interval (IdleSleep in DIConfig)	Database: Inventory	Process: EcDIActionDriver (DPAD)	The DPAD gets batches of actions (with status = null or status = RETRY) from the DIInsertActionQueue.

**Table 4.10-4. Data Pool ECS Insert CSCI Process Interface Events (2 of 4)**

Event	Event Frequency	Interface	Initiated By	Event Description
R3. Get file info	Once per file per ECS granule to be inserted	Database: Inventory (EcInDb) (AmDataFile/AmBrowseDataFile)	Process: EcDIActionDriver (DPAD)	The DPAD gets file name information for each file in the granule from the Inventory database.
R4. Get volume groups	Once per ECS granule to be inserted	Database: Inventory (DsStVolumeGroup)	Process: EcDIActionDriver (DPAD)	The DPAD gets the name of the open volume group for the granule's collection (shortname, versionid) from the Inventory database.
R5. Get tape info	Once per file per ECS granule to be inserted	Process: Java Quick Server Database: COTS StorNext metadata database	Process: EcDIActionDriver (DPAD)	The DPAD calls the Java Quick Server, which runs on the StorNext (COTS) metadata server, to retrieve tape label information for the granule files, based on the volume group information from the Inventory database.
R6. Update status	Once per ECS granule to be inserted	Database: Inventory (DIInsertActionQueue, DIActiveInsertProcesses)	Process: EcDIActionDriver (DPAD)	The DPAD updates the status of the insert in the AIM Inventory database.
R7. Invoke copy	Once per file per ECS granule	Process: EcDIQuickServer	Process: EcDIActionDriver (DPAD)	The DPAD chooses a QuickServer on an ECS Service Host to perform the file copy operation. The Service Host is chosen based on availability.
R8. Copy	Once per file per ECS granule	Storage Device: Data Pool disk (managed by COTS StorNext Storage Area Network) Storage Device: ECS Archive tape or cache (managed by COTS StorNext)	Process: EcDIQuickServer DIAdCopy	The QuickServer, running on an ECS Service Host, uses the DIAdCopy to copy the science file and its associated metadata xml file from the ECS Archive (tape or cache) to the Data Pool file system associated with the granule's collection.
R9. Invoke checksum	Once per file per ECS granule	Process: EcDIQuickServer	Process: EcDIActionDriver (DPAD)	The DPAD chooses a QuickServer on an ECS Service Host to perform the file checksum operation. The Service Host is chosen based on availability.

**Table 4.10-4. Data Pool ECS Insert CSCI Process Interface Events (3 of 4)**

Event	Event Frequency	Interface	Initiated By	Event Description
R10. Checksum	Once per file per ECS granule	Storage Device: Data Pool disk (managed by COTS StorNext Storage Area Network) Storage Device: ECS Archive tape or cache (managed by COTS StorNext)	Process: EcDIQuickServer	The QuickServer, running on an ECS Service Host, checksums the science file and the associated metadata xml file if needed on the Data Pool file system (temp directory).
R11. Extract band info	Once per ECS science granule, where the collection is enabled for HEG conversion (i.e., convertEnabledFlag is on for the collection)	Storage Device: temp directory in Data Pool file system	Process: EcDIActionDriver (DPAD) bandtool	The DPAD uses the bandtool utility to extract band information from the science granule, and writes band information to a temporary file in the Data Pool file system
R12. Invoke registration	Once per ECS science granule	Process: EcDIInsertUtility (NDPIU)	Process: EcDIActionDriver (DPAD)	The DPAD invokes an instance of the NDPIU from a pool to perform the granule registration.
R13. Get metadata	Once per ECS QA, PH, DAP and browse granule	Database: Inventory (EcInDb)	Process: EcDIInsertUtility (NDPIU),	The NDPIU gets metadata about the ECS QA, PH, DAP, brose granule from the Inventory database.
R14. Write xml	Once per ECS QA, PH, DAP and browse granule	Storage Device: temp directories in Data Pool file system	Process: EcDIInsertUtility (NDPIU)	The NDPIU writes the xml metadata file for the granule to the temp directory on the Data Pool file system.
R15. Register	Once per ECS granule	Database: Inventory	Process: EcDIInsertUtility (NDPIU)	The NDPIU populates basic tables in the AIM Inventory database with inventory information about the granule.
R16. Move science and xml to hidden directory	Once per data file and xml file per ECS granule	Storage Device: temp and hidden directories in Data Pool file system	Process: EcDIInsertUtility (NDPIU)	The NDPIU moves the science file(s) and xml file for the granule from the temp directory in the Data Pool file system to the appropriate hidden directory ( under /.orderdata).

**Table 4.10-4. Data Pool ECS Insert CSCI Process Interface Events (4 of 4)**

Event	Event Frequency	Interface	Initiated By	Event Description
[R17. Update status]	Once per insert request	Database: Inventory	Process: EcDIActionDriver (DPAD)	The DPAD updates the insert request status in the DIInsertActionQueue.
[P1. Insert action]	Once per publication request	Database: Inventory (DIInsertActionQueue)	Process: Data Pool Ingest Processing	The DPL Ingest Processing server places a request for granule publication in the DIInsertActionQueue.
[P2. Get actions]	Continuously, as long as there are actions in DIInsertActionQueue with status = null or status = RETRY. If no actions, once per configured time interval (IdleSleep in DIConfig)	Database: Inventory	Process: EcDIActionDriver (DPAD)	The DPAD gets batches of actions (with status = null or status = RETRY) from the DIInsertActionQueue.
P3. Invoke publication	Once per ECS granule to be published in the Data Pool	Process: EcDIInsertUtility (NDPIU)	Process: EcDIActionDriver (DPAD)	The DPAD invokes an instance of the NDPIU from a pool to perform the granule publication.
P4. Publish registered granule	Once per ECS granule to be published in the Data Pool	Database: Inventory	Process: EcDIInsertUtility (NDPIU)	The NDPIU populates additional tables in the Inventory database with inventory information needed to support web access to the granule.
P5. Move to public directory	Once per ECS granule to be published in the Data Pool	Storage Device: hidden and public directories in Data Pool file system	Process: EcDIInsertUtility (NDPIU)	The NDPIU moves the data file(s) and xml file for the granule from the hidden directory in the Data Pool file system to the appropriate public directory
P6. Update status	Once per ECS granule to be published in the Data Pool	Database: Inventory	Process: EcDIActionDriver (DPAD)	The DPAD updates the insert request status in the DIInsertActionQueue to the final request state, and removes the insert request from the DIActiveInsertProcesses table.

#### 4.10.4 Data Stores

There is one data store associated with the Data Pool subsystem. It is the Inventory database (AIM DB). Table 4.10-5 provides a description of these data stores.

**Table 4.10-5. Data Pool Data Stores**

<b>Data Store</b>	<b>Type</b>	<b>Description</b>
Inventory DB	Postgres	The Inventory (AIM) database implements the large majority of the persistent data requirements for the Data Pool subsystem. The Inventory database contains: a) inventory data for the Data Pool granules b) configuration data for the Data Pool; c) interim processing data for the Data Pool utilities; d) data for monitoring Data Pool insert queues and processing; e) Data Pool access statistics; and f) information about data pool entities such as collection groups, collections, file systems, compression algorithms, and themes.

## 4.11 Bulk Metadata Generation Tool Subsystem Overview

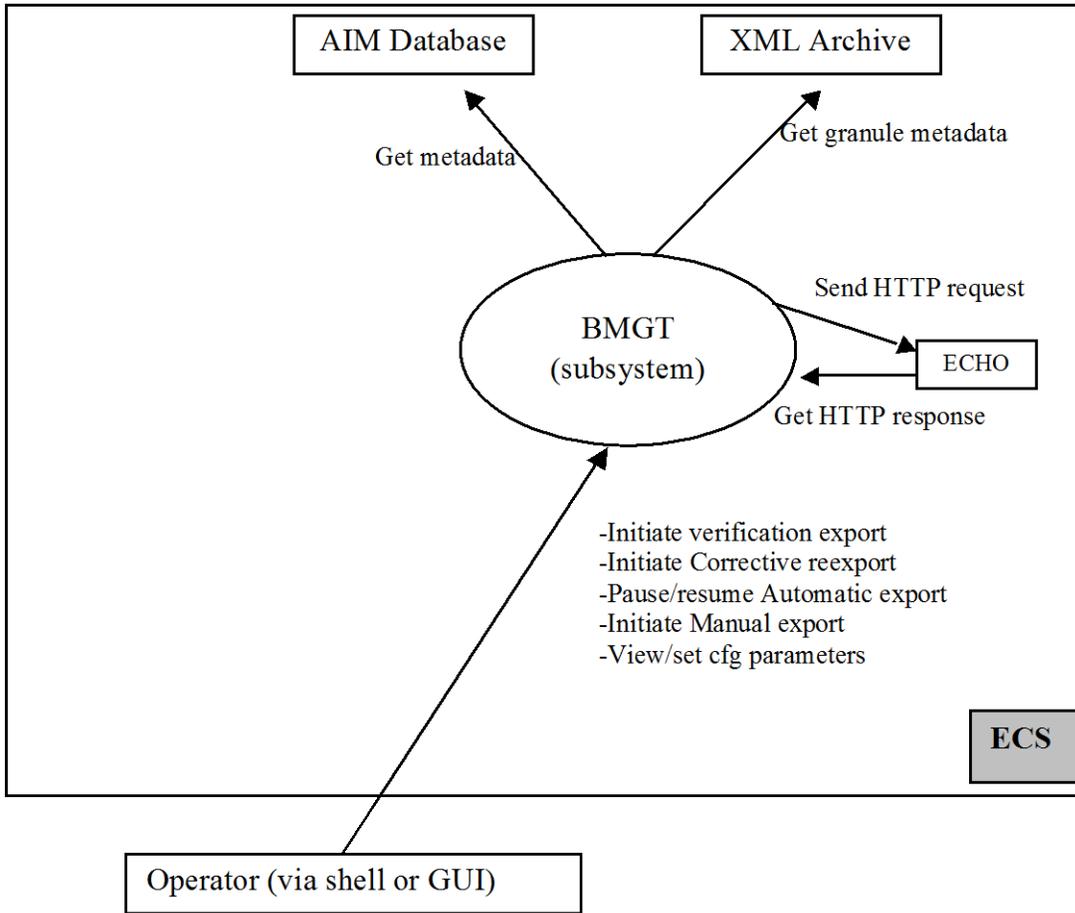
The ECS Bulk Metadata Generator Tool (BMGT) was created to support the development of value-added providers and external search and order tools by providing them with detailed metadata for the collections and granules archived at a DAAC. Currently, the EOS Clearinghouse (ECHO) is the only consumer of this capability. Since ECHO and Reverb are the primary search and order interface for ECS holdings, BMGT has a vital role in the ECS system. BMGT automatically exports metadata to ECHO as changes occur to the ECS data. Metadata can also be generated and exported on demand. There is currently no supported mechanism for the export of BMGT generated metadata to value added providers other than ECHO. Value added providers interested in this metadata should use ECHO's client APIs to access it.

Generally, BMGT metadata export is initiated at a regular interval by a polling server in order to export any changes to DAAC holdings in a timely manner. At the end of each interval, metadata reflecting added, removed, or changed granules and collections in the DAAC archive is generated in XML format. The generated products are exported to ECHO via HTTP PUT and DELETE requests where they are ingested into the ECHO catalog. In addition to automatically exporting metadata reflecting changes to the archive holdings, the BMGT can be manually executed to generate metadata for specific granules and collections. The output of these 'Manual' exports includes the products requested by the operator. Manual exports can be used to reconcile any discrepancies with ECHO or produce targeted sets of metadata for a particular task.

In addition to core granule metadata, retrieved from the XML met files in the ECS XML archive, BMGT exports additional metadata which may be useful to ECHO. This additional data includes the visibility of the granule, the URLs (if any) to immediately download the data (as well as metadata, Browse, and other ancillary files) from the datapool FTP/HTTP server, insert/update times, additional spatial metadata, etc.

### 4.11.1 BMGT Subsystem Context

Figure 4.11-1 is the BMGT Subsystem context diagram. The diagram shows the high level events generated between BMGT and other subsystems.



**Figure 4.11-1. BMGT Subsystem High Level Context Diagram**

Table 4.11-1 provides descriptions of the interface events in the BMGT Subsystem context diagram.

**Table 4.11-1. BMGT Subsystem High Level Interface Events**

Event	Interface Event Description
Initiate Corrective re-export	The operator initiates a script, which will initiate the generation of a BMGT package which contains the metadata for granules and collections which were queued for re-export due to an error returned from ECHO.
Initiate Verification export	The operator initiates a script, which will enqueue requests for the export of verification metadata for the granules and collections specified by the operator. This metadata will be used to verify that the metadata in ECHO's catalog matches that in ECS.
Pause/resume Automatic export	The operator pauses or resumes the automatic export of metadata via the BMGT GUI. The Automatic driver server must also be running in order to populate the export queue.
Initiate Manual Export	The operator runs a script, which will enqueue requests for the export of metadata for the items requested by the operator i.e., based on a list of granules/collections/groups and desired products, rather than an event time range.
Get metadata	BMGT reads collection, granule, browse, and other metadata from the AIM database.
Get granule metadata	BMGT reads granule/collection metadata from the XML/ODL metadata files located on the small file archive.
Send HTTP request	BMGT sends an HTTP PUT (for an insert update) or DELETE (for a delete) request to ECHO.
Get HTTP response	BMGT receives a synchronous response to the HTTP PUT or DELETE, detailing the result of the request, whether successful or otherwise.
View/set cfg parameters	The operator uses the BMGT GUI to view the current values for various configuration parameters that affect the behavior of BMGT. If the operator is logged in to the GUI as a read/write user, he/she can modify these values. The Operator can also view the group configuration and verification status.

#### 4.11.2 BMGT/ECHO Interface

The interface from BMGT to ECHO ('Send HTTP request' in Figure 4.11-1) is a REST style interface, meaning that all interaction is via standard HTTP requests (in this case PUT and DELETE requests). In the case of a granule or collection insert, a PUT request is sent, the body of which contains the granule or collection metadata. In the case of a deletion, a DELETE request is sent with not body. The URL to which the request is made indicates the identity of the affected resource. The interface is idempotent in that multiple inserts of the same item with the same metadata will have the same effect as a single insert, and multiple deletions of the same item will result in the desired effect of the item no longer existing in the system. This allows BMGT to greatly simplify its event handling, as redundant inserts or deletions of non exported items are legal (even if not overly efficient).

The interface from ECHO to BMGT ('Get HTTP response' in Figure 4.11-1) is in the form of a synchronous response to the HTTP PUT/DELETE request. The HTTP status code as well as the

contents of the response body, indicate whether the request was successful or whether it encountered errors.

Table 4.11-2 and 4.11-3 describe the possible contents of the HTTP requests/responses which make up the ECHO/BMGT interface. *Note: all schema files referenced below can be found at <http://www.echo.nasa.gov/ingest/schemas/operations/> (unless otherwise specified)*

**Table 4.11-2. Request Content Types**

File Type	Schema	Description
METC	Collection.xsd (or a provider specific ISO schema)	Collection metadata.
METG	Granule.xsd (or a provider specific ISO schema)	Granule metadata.

**Table 4.11-3. Response Content Types**

File Type	Schema	Description
Ingest Summary Report	EchoRestExceptions.xsd ( <a href="https://api.echo.nasa.gov/echo-rest/wadl/bindings/EchoRestExceptions.xsd">https://api.echo.nasa.gov/echo-rest/wadl/bindings/EchoRestExceptions.xsd</a> )	Listing of errors, if any, encountered during the fulfillment of the request.

### 4.11.3 ECS Events and BMGT products

In general, BMGT is run automatically on a polling interval. This interval can be any interval desired by the DAAC, from seconds to hours, but in general it is best to be on the order of 10 to 60 minutes. At the end of each interval, BMGT will enqueue for export to ECHO (via the interface described in 1.1.2) metadata reflecting any relevant changes to the ECS holdings since the end of the last interval. Relevant changes include inserts, deletes, and updates to collections and granules which are configured to be exported to ECHO. Such events cause database triggers to be fired which record the events in database tables where they are then picked up by BMGT. On each polling interval, BMGT will select up to a configured number of events which are not yet enqueued for export, and enqueue the export of the current state of the affected object. If multiple events for the same item are picked up at the same time, only a single request will be enqueued. Since the current state (at the time of request processing) will be used to determine the type of export (insert or delete), it does not matter what type of event caused the request to be enqueued, or even if the events contradict one another (e.g. an insert and a deletion of the same item). Table 4.11-4 describes the types of events which are relevant to BMGT/ECHO, and what is exported to reflect each event during an automatic export. In a Manual or Verification BMGT export, export requests are similarly enqueued, but for items specified by operator input rather than by events. By default, Manual BMGT exports will export the current state of the requested item, as automatic export does. However, it is possible to request, via command line options, the export of inserts or deletes for only those specified items which are indeed inserted or (logically or physically) deleted, and in special cases, to force the delete of items which would normally be eligible for insert.

**Table 4.11-4. ECS Event to BMGT Product Mapping (1 of 2)**

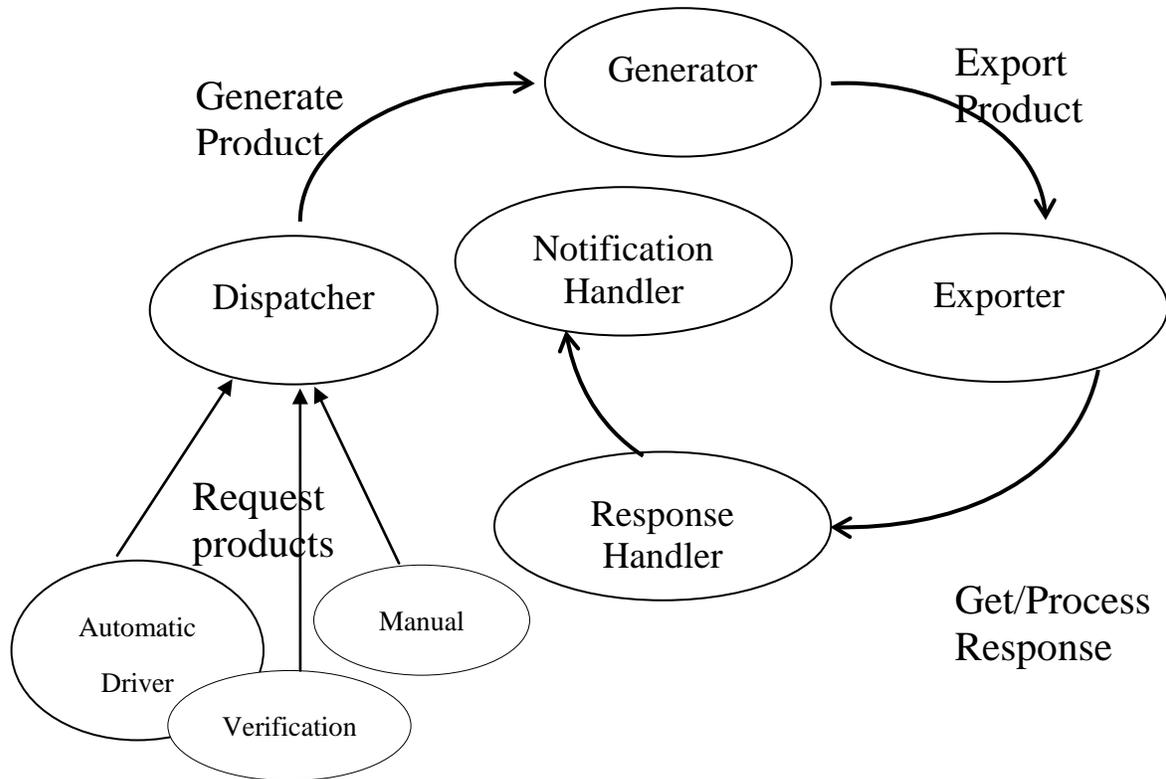
Event Type	Cause/location	Products [<FileType> <Schema element>]	Sample URL
GRINSERT	Granule inserted into AIM database, or logically undeleted.	Granule PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
GRDELETE	Granule deleted from AIM either logically or physically.	Granule DELETE	DELETE <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
GRUPDATE	Granule core metadata updated in AIM/XML file.	Granule PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
GRHIDE	Granule hidden in AIM DB, and not available for order.	Granule PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
GRUNHIDE	Granule unhidden in AIM DB, and now available for order.	Granule PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
CLINSERT	Collection inserted into AIM DB.	Collection PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/datasets/MODIS%2FAqua%20Sea%20Ice%20Extent%20Daily%20L3%20Global%201km%20EASE-Grid%20Day%20V086">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/datasets/MODIS%2FAqua%20Sea%20Ice%20Extent%20Daily%20L3%20Global%201km%20EASE-Grid%20Day%20V086</a>
CLDELETE	Collection deleted from AIM DB.	Collection DELETE	DELETE <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/datasets/MODIS%2FAqua%20Sea%20Ice%20Extent%20Daily%20L3%20Global%201km%20EASE-Grid%20Day%20V086">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/datasets/MODIS%2FAqua%20Sea%20Ice%20Extent%20Daily%20L3%20Global%201km%20EASE-Grid%20Day%20V086</a>

**Table 4.11-4. ECS Event to BMGT Product Mapping (2 of 2)**

Event Type	Cause/location	Products [<FileType> <Schema element>]	Notes
CLUPDATE	Collection modified in AIM DB.	Collection PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/datasets/MODIS%2FAqua%20Sea%20Ice%20Extent%20Daily%20L3%20Global%201km%20EASE-Grid%20Day%20V086">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/datasets/MODIS%2FAqua%20Sea%20Ice%20Extent%20Daily%20L3%20Global%201km%20EASE-Grid%20Day%20V086</a>
BRINSERT	Browse file linked to science granule in AIM DB	No Export	
BRDELETE	Last link to browse file deleted in AIM DB	No Export	
BRLINK	Science/Browse link added to AIM DB.	Granule PUT (for linked granule)	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
BRUNLINK	Science/Browse link removed from AIM DB.	Granule PUT (for linked granule)	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
QAUPDATE	QA parameters modified in AIM DB.	Granule PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
GRURLINS	Granule Added to public datapool.	Granule PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
GRURLDEL	Granule logically or physically removed from public datapool.	Granule PUT	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>
CLMOVED	Collection metadata updated in DPL DB.	Granule PUT (for all public granules in the collection)	PUT <a href="https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535">https://api-test.echo.nasa.gov/catalog-rest/providers/NSIDC_TS1/granules/SC%3AMYD29P1N.006%3A6658535</a>

#### 4.11.4 BMGT Architecture

Figure 4.11-2 displays the BMGT Architecture diagram.



**Figure 4.11-2. BMGT Architecture Diagram**

Table 4.11-5 provides descriptions of processes shown in the architecture diagram.

**Table 4.11-5. BMGT Processes (1 of 2)**

Process	Type	Hardware CI	COTS/ Developed	Functionality
Automatic Driver	Server	OMSHW	Developed	Server process polls the AIM event History table to find any unexported events which are eligible for export. Events for the same item (collection or granule) are consolidated. Requests are added to the BMGT export request table pursuant to the selected events and the events are marked as having been picked up.
Manual Driver	Application	OMSHW	Developed	The operator runs a script to explicitly tell the BMGT to enqueue export requests. The Manual start script provides a large number of options for generating the export package to fit the operator's needs. These options include the ability to release any corrective export requests awaiting operator approval.
Verification Driver	Application	OMSHW	Developed	The Operator runs a script to request the export of full metadata for a specified list of granules and collections for the purpose of verifying and reconciling any differences between ECS and ECHO catalogs. Alternatively, the Operator can request the query from ECHO of a list of granules or collections to be compared against the AIM holdings to determine if there are any extra or missing items.

**Table 4.11-5. BMGT Processes (2 of 2)**

Process	Type	Hardware CI	COTS/ Developed	Functionality
Dispatcher	Server	OMSHW	Developed	The server polls the BMGT export request table to find any requests on any of its logical queues awaiting export. It then adds these requests to one of its internal queues (adding an activity to the BMGT export activity table), and then works off the queues, generating, exporting, and handling responses for each one. <b>NOTE: Each of the remaining 'server' components actually run as parts of the dispatcher server, but are listed here because they each have a logically separate function.</b>
Generator	Server	OMSHW	Developed	Once an export activity has been picked up off the queue for export, it is passed to the generator, which finds the current state of the item, and then generates the appropriate metadata based on the item state and the requested export type. If the appropriate or requested action is a delete, then no metadata is generated. Otherwise, the native metadata is processed into the appropriate format, and additional dynamic metadata elements are added as appropriate based on the current configuration.
Exporter	Server	OMSHW	Developed	Once metadata is generated pursuant to an export request, the dispatcher calls the exporter to send that metadata to ECHO. If the export is a delete, an HTTP DELETE is sent with no body. Otherwise, a PUT is sent containing in its body the item's metadata. The URL to which the PUT or DELETE is directed indicates the item being inserted, deleted, or updated. If the export is on behalf of a verification request, then the request will contain a special query parameter which tells ECHO to invoke special verification handling.

EBIS document 920-TDx-001 (HW Design Diagram) provides descriptions of the HWCIs and document 920-TDx-002 (Hardware-Software Map) provides site-specific hardware/software mapping.

#### **4.11.5 Use of COTS in the BMGT Subsystem**

- JRE

The JRE constitutes the Java virtual machine and the Java platform core libraries. It provides applications with the Java platform. Included with it is JAXP (Java API for XML Processing), which is also used by BMGT.

- Hibernate

Hibernate provides access to the postgres database, including the ability to link java objects to database records by way of annotations.

- JDOM

JDOM libraries allow java applications to create and edit xml documents.

- Postgres Server

The BMGT accesses the Inventory database to read inventory metadata.

- JAF / Javamail

Java Activation Framework (JAF) and Javamail provide BMGT the capability to send email messages.

- c3p0

Used to maintain a pool of connections to the Inventory database.

- Jersey

A java library for setting up REST style web services.

- Dojo

A java script library for building dynamic web GUIs.

#### **4.11.6 BMGT Subsystem Software Description**

##### **4.11.6.1 BMGT CSCI Functional Overview**

The BMGT Dispatcher server is always running, waiting for work to do. However, metadata generation will only be initiated when one of the Drivers enqueues export requests. There is one driver for each BMGT cycle type (EVENT, MAN, CORR, verification [VER, SHORT, and INCR]). Each driver is responsible for deciding which granules or collections to enqueue requests for, as well as what type of export to request. VER, INCR, SHORT, CORR, and MAN pre processors are actually the same executable, the Manual Driver, but are called with different

options in each case. The Manual Driver is a stand-alone executable which is called interactively by the operator, while the Automatic Driver (which enqueued EVENT requests) is a server process which polls for new events on the inventory event queue for which to enqueue requests.

In general, the use case for each BMGT cycle type is as follows:

Automatic - Automatic Driver is a server process configured to poll on a set interval for new events in the AIM event table for which export requests must be enqueued. These events are then added to the EVENT queue and marked as processed in the AIM event table.

Manual - The operator runs the Manual Driver to explicitly request the export of specified collections and/or granules. Requests are added to the MAN queue pursuant to the specified options. The driver provides a large number of options for specifying the exact items and export type to fit the operator's needs.

Corrective - The operator runs the manual driver with the --corrective option to release any export requests which have been blocked. Blocked requests are those which were automatically enqueued due to errors, or those which have themselves encountered an error which requires operator intervention.

Short - The operator runs the Manual Driver with the --short option to initiate the request of a listing of all granules in a collection or all collections followed by the comparison of this listing against the local database. Any discrepancies will result in the addition of new export requests to the CORR queue.

Long Verification - The operator runs the Manual Driver with the --long option to enqueue requests on the VER queue for the export of the specified granules and/or collections for the purpose of full comparison with the metadata on record in ECHO.

Incremental Verification. The operator runs the Manual Driver (or it is run via a cron) with the --incremental option to select and add to the INCR queue a set of the least recently updated granules (and their associated collections) which have not yet been incrementally verified in this manner since their last update. This metadata is compared against that on record in ECHO.

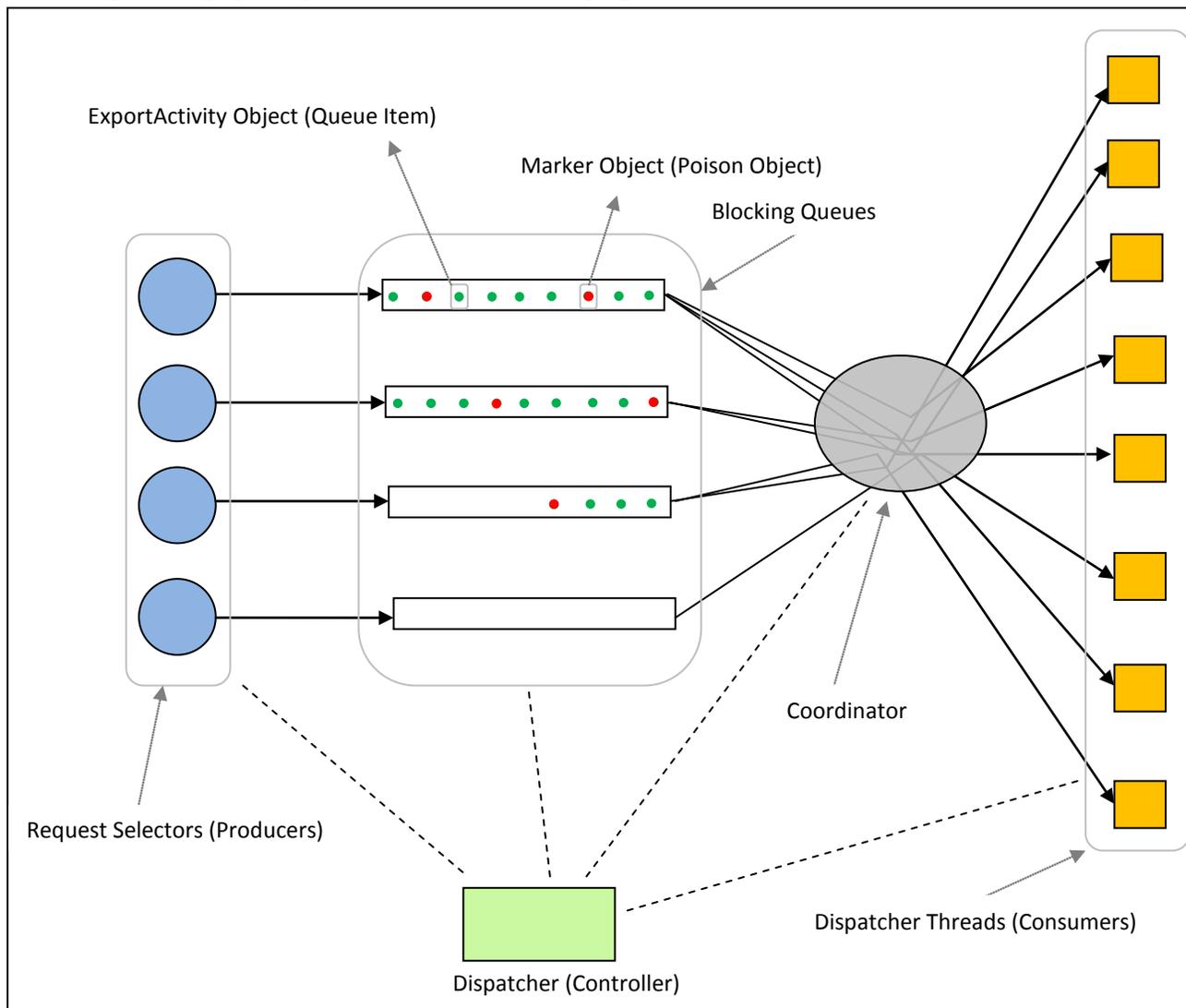
Once an export request has been enqueued, it will be eligible for dequeuing by the BMGT Dispatcher. The dispatcher will poll for requests on the queue and pass them to the Generator, which will generate the appropriate metadata. The Dispatcher will then pass the metadata to the Exporter which will perform an HTTP PUT or DELETE to ECHO to propagate the desired insert, update, or delete. The HTTP response will then be processed by the Response Handler to determine whether the export was successful or if there were errors and then processes any errors accordingly.

The following sections provide more detailed descriptions of the various components briefly described above.

### 4.11.6.2 Dispatcher

The Dispatcher is the central component of BMGT outside the database. It is responsible for orchestrating rest of the components of the system and setting the overall pace of exports to ECHO. On one hand, it communicates with database to look for new entries waiting to be exported to ECHO and on the other hand it communicates with Generator, Exporter and Response Handler which do the actual job of generating the metadata, sending it to ECHO and finally handling the response received from ECHO. Dispatcher holds the main controls of BMGT which are used to start, stop, suspend or resume exports to ECHO.

Figure 4.11-3 shows at a high level how the Dispatcher operates. The design of Dispatcher is based upon the popular producer-consumer design pattern.



**Figure 4.11-3. Dispatcher Using Producer-Consumer Design Pattern**

## Dispatcher Subcomponents

The Dispatcher is made up of 5 sub components:

- 1) Request Selector – Producer.
- 2) Blocking Queue
- 3) Dispatcher Thread- Consumer.
- 4) Coordinator
- 5) Dispatcher – Controller

The sub components are explained in detail below.

### Request Selector

This component is responsible for polling the `bg_export_request` table in the database for entries which are available for export to ECHO. This table is mapped to the `ExportRequest` data model class, which will be the dispatcher's interface to the table. The table will be referred to throughout this section as the `ExportRequest` table. The records in the `ExportRequest` table can be created by four different types of event drivers – event, manual, incremental and corrective. Records are also added to the table when a new collection is enabled for export. A field within this table called, `ExportQueue`, has six possible values – `EVENT`, `MAN`, `VER`, `INCR`, `CORR` & `NEW`. The table can be seen as holding records for six different logical queues one corresponding to each of these values. Each of the drivers feeds into one or more of these queues. An instance of Request Selector, from now on referred to as a “producer”, is created corresponding to each of the six database queues. The six producers run concurrently and can be halted or resumed independently. The “produce” operation of the producer corresponds to fetching records from the corresponding database queue and creating a “bucket” of `ExportActivity` objects (described below), each item in the bucket corresponding to one `ExportRequest` record fetched from the database. Only those records are fetched whose “Status” field is set to “PENDING”. A status of “PENDING” indicates that the metadata for a granule or collection corresponding to the record is recently inserted, updated, removed, enabled for export, or that a manual or verification export has been requested by the operator, but the changes are not yet sent to ECHO. An operator configured number of records are selected at a time. The status of all the selected records is updated to “STARTED” once the corresponding `ExportActivity` object is created.

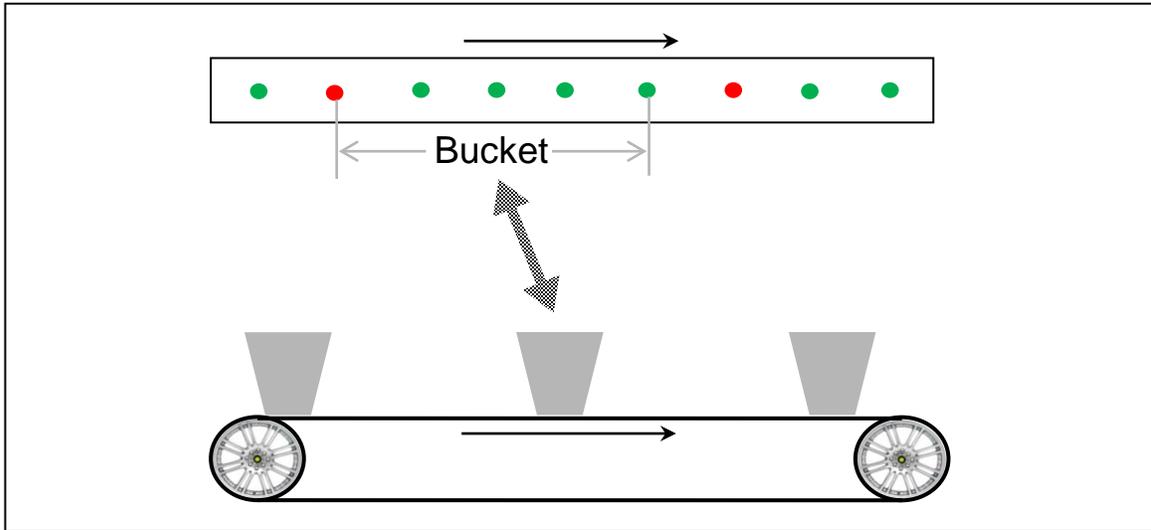
The `bg_export_activity` table in the database keeps track of the current export attempt corresponding to a record in `ExportRequest` table. This table is mapped to the `ExportActivity` data model class, which will be the dispatcher's interface to the table. The table will be referred to throughout this section as the `ExportActivity` table. Multiple export attempts could be made for a given entry in the `ExportRequest` table (for instance, if an export attempt fails and the request must be retried). A record is added into `ExportActivity` for each of the requests picked up by the dispatcher. `ExportActivity` has a field, called “Status”, which is changed progressively within the Dispatcher Threads (consumers) as different components work on it until it reaches a

terminal status. The ExportActivity objects are stored in a buffer as they are created. The contents of a single database fetch which are put into the buffer will be referred to as a “bucket” (see figure 4.11-4). Along with the ExportActivity object, producer also adds a “marker object” at the end of the bucket. The marker object does not contain any useful data itself, but is a signal to the consumer that the end of the bucket is reached. Note that the actual size of the bucket will be one more than the size of the database fetch. The producer puts all the contents of the bucket into a Blocking Queue (described next). The contents of the buffer are placed one at a time on the Queue until the buffer is empty. The producer has three different modes based on the operation that it is performing at any given moment. The mode when it is inserting the objects into the queue from the buffer will be referred to here as “Inserting” mode.

Once the objects are inserted, it goes back to checking for new entries in the database queue. This mode will be referred to as “Fetching” mode. If no entries are found, it goes into “Polling” mode. In the Polling mode, it periodically checks the availability of new entries until they are found.

## **Blocking Queue**

As described above, producers put the contents of a bucket into a queue. A single queue is associated with each producer. The queue is an Array Blocking Queue of fixed size which is shared by producers and Dispatcher Threads (described below), also referred to here as consumers. The producers insert data into the queue one object at a time. If the queue is full, the producer waits on the queue until space becomes available and the object is inserted. After the object is inserted, it tries inserting the next object in the bucket and so on until the bucket is empty. When all the objects in the producer’s bucket are inserted into the queue, it is free to produce a new bucket. The buckets on the queue are analogous to goods transported on a conveyor belt in an assembly pipeline (See Figure 4.11-4). For maximum throughput, the size of the queue will be sufficiently large so that the producer can produce a new bucket even before the consumers can consume the contents of the entire queue. That way, consumers don’t have to wait for the producer to produce a new bucket. Similarly, the size of the bucket will be chosen appropriately. If the size of the bucket is too small, it leads to frequent database requests and if it is too big, it requires more memory and it takes longer time to produce the bucket and consumers could be waiting (if the queue is small) while the producers produce the bucket.

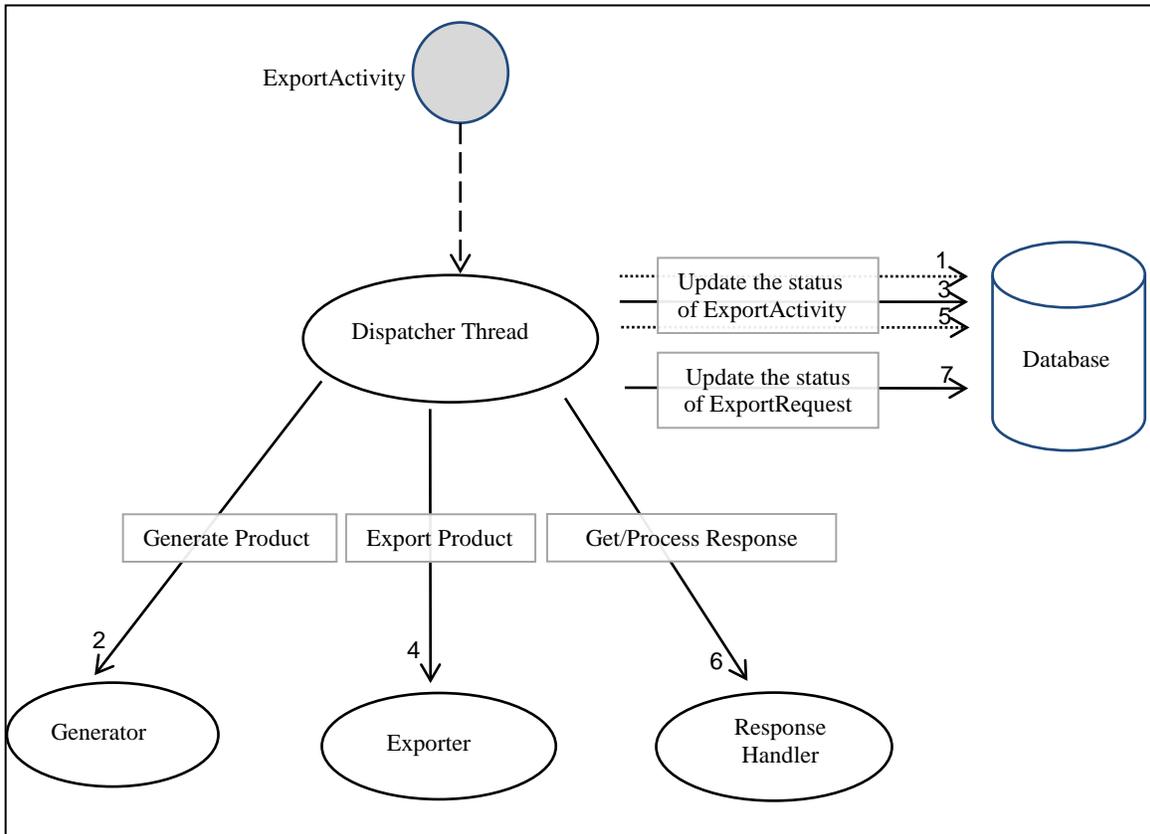


**Figure 4.11-4. Bucket on a Queue**

### **Dispatcher Thread**

Dispatcher Threads (or consumers), with the help of other components of BMGT, do the core job of generating and exporting the metadata to ECHO and handling the response received from ECHO. A Dispatcher Thread carries out all these operations synchronously in a single thread. But several (a configurable number) threads will be running concurrently to speed up the exports

Each consumer object is associated with exactly one instance of Exporter, Generator and Response Handler. The consumer passes the ExportActivity Object received from a queue and passes it to each of the components in a sequence and updates the status field of the ExportActivity object as it proceeds (see the figure 4.11-5). The contents of the ExportActivity Object are modified by the components as it moves from one component to another.



**Figure 4.11-5. Dispatcher Thread and the Sequence of Calls to Various Components**

- 1) Dispatcher Thread first sends the ExportActivity object to Generator requesting it to generate the metadata corresponding to granule or collection associated with the object. The status of the ExportActivity object is updated to “GENERATING”.
- 2) The generator generates the metadata (See Generator section) and populates the ExportActivity with a reference to an XML object which contains the metadata (or no metadata, if the item is deleted). The Dispatcher Thread itself is indifferent to the actual contents to be exported or the kind of export. For example, it does not treat a metadata INSERT any different from DELETE.
- 3) Once the Generator returns with the updated ExportActivity, it is sent to Exporter for exporting to ECHO. The status of ExportActivity is updated to “EXPORTING”.
- 4) The Exporter sends the request to ECHO and receives the response (see Exporter section) which is populated into the ExportActivity object.
- 5) The Dispatcher then submits the updated ExportActivity object to the Response Handler for evaluating the response (See Response Handler section). The status of ExportActivity is updated to “EVALUATING”.

- 6) Response Handler is responsible for setting the terminal status of the Export Activity and taking any necessary action based on the response received from ECHO. It can for example, mark the export entry for subsequent export attempt or send notification to the operator regarding errors or warnings which need operator's attention.
- 7) After the Response Handler returns, the Dispatcher Thread updates the status of ExportRequest objects to one of the terminal statuses if ExportRequest is completed or to a retry status if another attempt needs to be made to process the ExportRequest (Please see Response Handler section for the details regarding the terminal statuses).

## **Coordinator**

Coordinator is responsible for assigning queues and associated buckets to each of the consumers. Each bucket is consumed by one or more consumers at any given moment. The retrieval of entries in the bucket by the consumers is synchronized on the bucket and ensures that each consumer gets a distinct entry to process. If a consumer receives a regular ExportActivity object, it processes in the manner described under "Dispatcher Thread". But if a consumer receives a marker object instead of the regular ExportActivity object, it indicates that the bucket is consumed. Note that exactly one consumer of the bucket will receive the marker object. The marker objects mark the boundaries of a bucket. On receiving the marker object, it signals coordinator of the end of the bucket on the corresponding queue. Each of the consumers which are associated with a bucket will ask the coordinator for the status of the bucket before reading an item in the bucket in order to check if the bucket is a new bucket or is the same as they started with. If the bucket is different, the consumer goes back to the coordinator for a new bucket to be assigned. In summary, a consumer works on a single bucket on a queue until it is empty before changing the bucket. The new bucket assigned could be on the same queue or a different queue. But not all consumers which consumed a bucket exit the queue after the bucket is empty; one consumer, the one which actually found the marker object remains with the queue (and the rest of the consumers return to coordinator for a new bucket/queue). If the remaining consumer finds a new regular entry, it starts working on the new bucket as a "regular consumer", but if it does not find any entry, it goes into a polling mode and checks periodically for a new bucket on the queue. It also signals the coordinator that the queue is empty in the latter case. The coordinator will now not assign the queue to any new consumer when the request for new bucket is made. If the "polling consumer" finds a valid entry again, it signals the coordinator that the queue is non-empty and the bucket on the queue is ready to be consumed. The coordinator can now assign more consumers to the bucket. The "polling consumer" will switch to become a "regular consumer" on the same queue. If all the queues are empty, all the "regular consumers" will wait on the coordinator while each of the "polling consumers" poll.

When a consumer requests the coordinator for a new bucket on a queue, it selects the bucket based on a "fairness policy". Coordinator has a list of the number of consumers assigned to each queue in the "steady flow state". In the "steady flow state" none of the queues are empty for a sufficiently long time. The number of consumers which are assigned to a queue in the "steady flow state" is an operator configurable parameter. The parameters are chosen so that buckets on

all the queues are consumed evenly without one queue being assigned all the consumers. If a queue is empty though, the consumers which would have consumed a bucket on the queue in the “steady flow state” can be now be assigned to other queues so that the consumers are not wasted when the queue is empty. So the actual number of consumers consuming a bucket on a queue at any given time can be more than the configured number (which would be used in the “steady flow state”). The coordinator maintains a count of all the consumers assigned to each of the queues at any given moment. The coordinator uses this count and the configuration parameters to calculate the “fairness policy”. The coordinator will assign to a consumer, requesting a bucket to work on, a bucket of the queue which is 1) non-empty and 2) has the least percentage of consumers assigned to it compared to what it would have been assigned in the “steady flow state”.

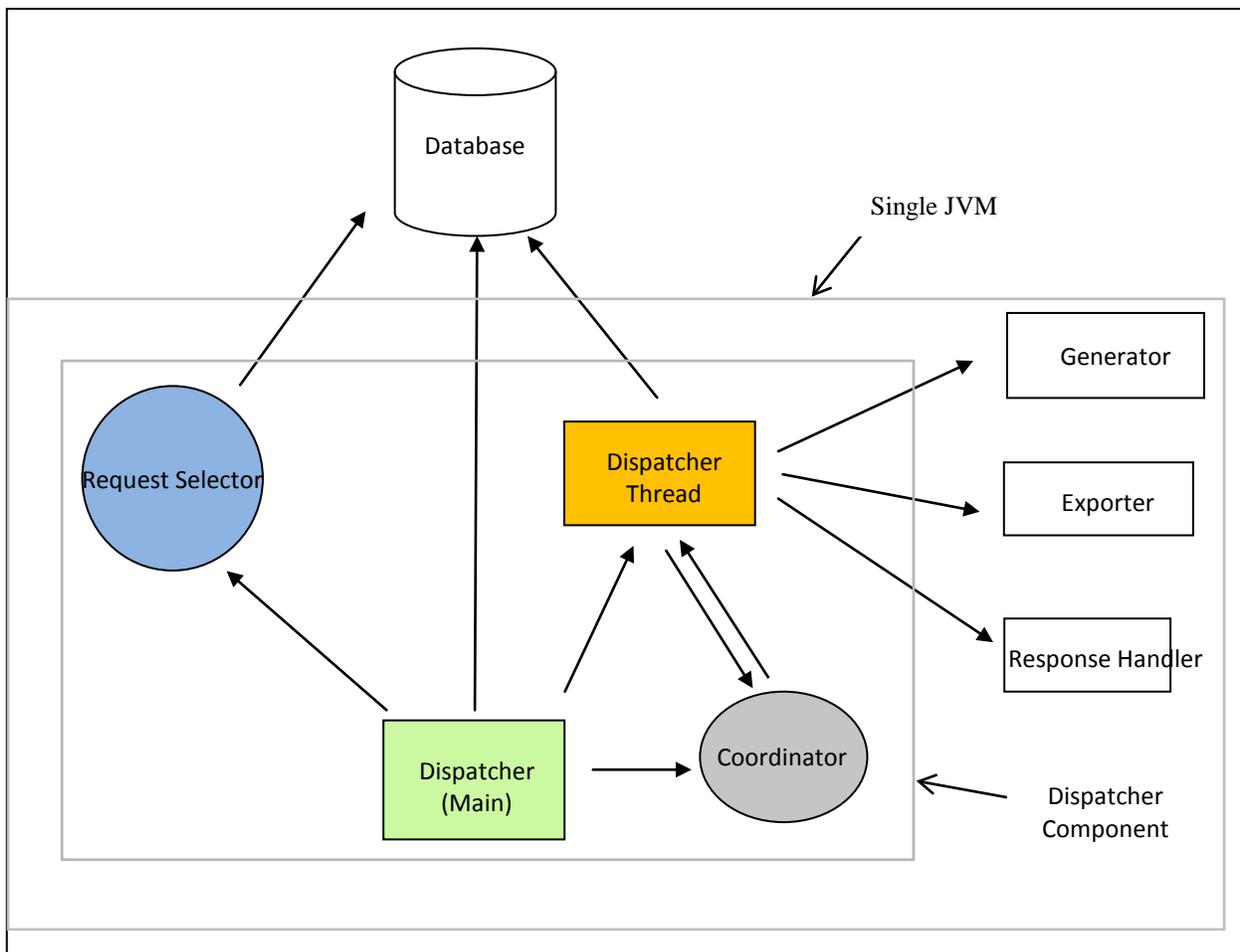
## **Dispatcher**

This component is responsible for controlling the rest of the components of the Dispatcher including start, stop, suspend and resume operations of the consumer-producer pipeline. This sub-component will also be referred to here as “controller” in order to avoid confusion with the top level component with the same name. Controller will have public interfaces to carry out the operations. These interfaces can be called from within GUI so that the operator can control Dispatcher (note, the interface to the GUI may be via signal entries to the database, rather than a direct RPC or web service call). When the operator issues a stop request on the controller, it is important that all the exports which are in the process are completed before stopping. In order for this to happen, controller will first stop producers so that no more buckets are fetched and wait for the consumers to consume all the buckets which are already created. The same will also happen when controller is paused. But when stopped, consumers and producers will be discarded whereas during pause, the objects will not be discarded, but wait for the controller to call resume. If the controller goes down unexpectedly, the ExportRequest and ExportActivity tables will have records whose status implies that the corresponding granules or collections are in the process of being processed when they actually are not. To rectify this, on startup, Dispatcher will check these tables for the inconsistent records and their status will be reverted back to the original status. Note that all the export requests to ECHO are idempotent and resending a request, which has already been sent but the status not changed due to an unexpected crash, does not cause any inconsistency or error (besides possible messages indicating that an item has already been deleted, which will be ignored).

It will be possible to change some of the configuration parameters even while the Dispatcher is running like changing the bucket size or polling frequency for new Requests. These parameters will be referred to as dynamic configuration properties, for changing other configuration parameters, queue size for example, start of Dispatcher will be required. Each of the producers and consumers will have their own controls which will be called by controller. The operator will be able to incorporate the changes to dynamic configuration properties without actually restarting Dispatcher (or BMGT), controller will do the necessary work to gracefully permeate the new parameters to various components of Dispatcher.

## Other Design Issues

The Dispatcher will be implemented as a Java Daemon process with its own main method. It will be started as part of the normal ECS mode start up process and will run continuously. The Dispatcher will pull in the jar files containing the Generator, Exporter, and Response Handler, as well as the data model classes (for database access). It is important to note that all the other components referenced here, (Generator, Exporter, etc.) could be considered as part of the Dispatcher, rather than separate top level entities. However, since they are logically separate, they will be implemented in separate java packages (in separate jar files) and will be referred to for all intents and purposes as separate entities. Dispatcher will also have a shutdown hook which would allow the operator to stop Dispatcher from command line.

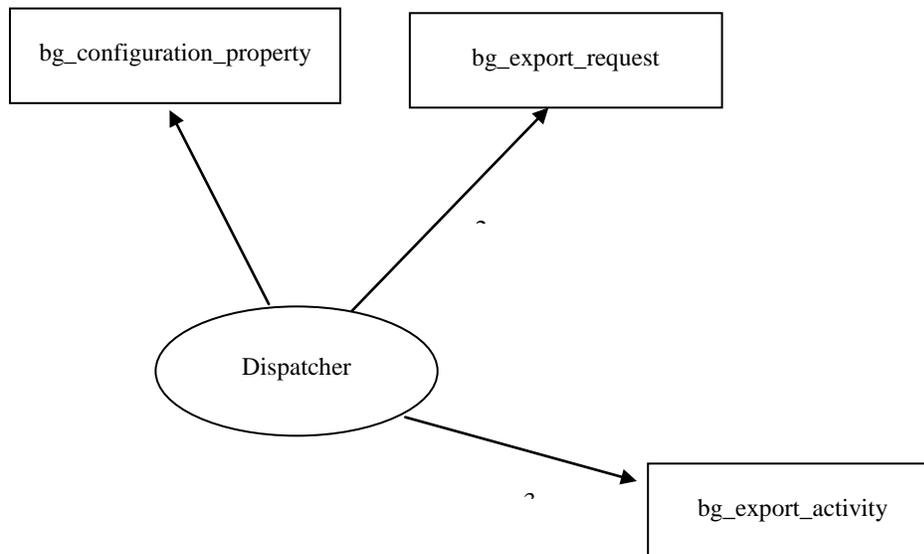


**Figure 4.11-6. Component Dependencies**

Also, note that it is not necessary that a consumer is associated with the same triplet of Exporter, Generator and Response Handler objects all the time. Instead, a pool of each type of objects can be initialized and a consumer can retrieve these objects on demand using a factory method present within these components and return it back to the pool when it is done using it. The object can be reused by other consumers later on or it can be discarded if idle for sufficiently long time. This corresponds to the object pool design pattern. This can lead to less memory usage as not all the objects of each component will spend same amount of time on an ExportActivity object and one component will be waiting while another component works on it. When the number of consumers is large, the system will, on an average, need less instances of a component which takes the least amount of time doing processing among the three. And the component which is slowest will require maximum number of instances, not greater than the number of consumers. If it is known that a particular component is going to be the slowest, it can as well be tightly attached to the Dispatcher Thread.

## Workflow

High level diagram of Dispatcher's interaction with the database.



**Figure 4.11-7. Dispatcher Database Sequence**

1. Read system configuration
2. Poll for eligible export requests.
3. Insert export activity and link to the export request. Process the activity, updating its state as it proceeds.

### 4.11.6.3 Generator

#### Overview

The Generator is responsible for confirming the export type of a request (correcting if necessary), determining whether metadata generation should take place, reading and translating metadata from the ECS archive into the appropriate format and, finally, calculating and inserting metadata elements based on the ECS catalog record and external configuration as appropriate.

Generator can produce collection or granule metadata in one of two types, ECHO10 or ISO/SMAP, meaning there are four distinct metadata types: Echo10Collection, Echo10Granule, SmapCollection and SmapGranule. Information from the configuration record for the collection associated with the activity determines the appropriate output format.

While there are no concrete plans for more metadata types, it is certainly conceivable that they would be added at a later point, for instance, if future missions provide metadata conforming to a different ISO profile. Designing the Generator to seamlessly handle this eventuality would be over-engineering, and would increase the cost of implementation. However, the Generator is designed with consideration of reducing the impact if and when such enhancements are necessary. The generator makes use of polymorphism methodologies, for instance using abstract classes for root metadata classes such that new classes can be written which extend these abstractions and fairly easily plugged in to the Generator.

#### Architecture

The main responsibility of the Generator subsystem is to produce and populate an XML representation of a metadata object. Before doing so, Generator must also determine the appropriate export type for the request, and whether generation should be attempted.

Input sources required by Generator fall into four categories: configuration properties, resource files, request information and metadata information. Configuration properties and resource files are read only once, into the fields of a singleton object, from which representations of these data can be safely obtained by individual Generator instances on separate threads. Information about the context of a request (such as, say, whether the collection it refers to exists in the database tables) is obtained either by querying the database directly via a data access object (DAO), or from the persisted request object itself, both of which reside in the BMGT Common package. All calls to the DAO are carried out in a single block, so that only one database session is required.

Once the necessary information about the request is obtained, Generator decides which of five actions to take:

- a. Make no changes to the ExportActivity object
- b. Change the export type to DELETE
- c. Throw a GenerationException, causing Dispatcher to skip exporting this request

- d. Change the export type from OPEN to ADD, attempt to generate the metadata, and if successful, populate the generatedMetadata field of the ExportActivity.
- e. Attempt to generate the metadata, and if successful, populate the generatedMetadata field of the ExportActivity (no change to export type).

The Generator will always choose the export type and execution path that matches the current state of the specified item, even if that contradicts the type of the event which originally caused the generation. If an item exists, and is eligible for export, its full metadata will be generated (resulting in an insert or update to ECHO). Otherwise, a delete will be generated. There are two exceptions to this rule. If a “full collection update” is requested as part of a manual export, indicating that a collection needs to be completely removed and re inserted into ECHO in order to propagate an update—or, if a “force delete” is requested when a collection is not deleted—then the generator will return a deletion response to the caller, despite the collection not truly being deleted.

Generator can produce collection or granule metadata in one of two formats, ECHO10 or SMAP, meaning there are four distinct metadata types: Echo10Collection, Echo10Granule, SmapCollection and SmapGranule. Information from the configuration record for the collection associated with the activity determines the appropriate output format.

Native metadata formats map to output formats as follows:

**Table 4.11-6. Metadata Native to Target Schema Mappings**

<b>Native metadata format:</b>	<b>Output metadata format:</b>
ECS	ECHO 10
ISO/SMAP	ISO/SMAP

When ECHO10 is the expected output format, the native metadata file must first be translated from the ECS format into ECHO10 XML. This is done using an XSLT stylesheet and processor. In the case of Echo10Collection, native metadata exist in ODL descriptor files, which must be converted into ECS XML before translation. Once the ECHO10 XML has been obtained, it is unmarshaled into a JAXB representation of the metadata type. Additional metadata items, such as insert and update times, URLs, spatial and orbit metadata, cloud cover, browse linkages, and other elements are obtained or calculated, then inserted into the JAXB content tree. Finally, the completed metadata content is once again marshaled to XML.

When SMAP is the expected output format, the process is much simpler. The native metadata in this case are also in SMAP format, but lack a few elements or values (e.g. insert and last update times, granule URLs, DIFid). These are passed in as parameters using an XSLT stylesheet and processor. We chose to forego the added overhead of a JAXB content model for SMAP at this time, since the data required for insertion were so few and simple in structure.

## Generator Subcomponents

### Generator Core

The Generator class exposes generate(ExportActivity) to the Dispatcher. This method mutates ExportActivity, and is the access point to all of Generator's functionality. The generate method will follow one of three execution paths, depending on the system's state:

**Table 4.11-7. Generator Behavior**

<b>System state:</b>	<b>Behavior of generate(ExportActivity):</b>
ExportActivity.ExportType is FORCE_DEL	Do nothing and return.
ExportActivity.ExportType is DEL and collection or granule does not exist in the database.	Do nothing and return.
ExportActivity.ExportType is DEL and collection or granule exists in the database.	Throw GenerationException.
ExportActivity.ExportType is OPEN, but the collection or granule does not exist in the database.	Set ExportActivity.ExportType to DEL and return.
ExportActivity.ExportType is not DEL or FORCE_DEL, but item is not exportable.	Throw GenerationException.
ExportActivity.ExportType is OPEN and collection or granule exists in the database, and is exportable.	Set ExportActivity.ExportType to ADD and attempt to generate metadata.
ExportActivity.ExportType is ADD or anything not listed above	Attempt to generate metadata.

### Metadata model

An instance of a metadata model represents a single metadata entity, either a Granule or a Collection, providing an structure for storing and handling the XML representation (either String or JAXB content tree), and providing behaviors needed for obtaining and calculating metadata, as well as populating the XML representation. Note that these objects represent discrete metadata units, not metadata generators. This better fits ECHO's new REST API, in which there is one metadata item per request, and represents a design departure from BMGT 8.1.

ECHO10 metadata models represent XML using a JAXB content tree. JAXB was chosen because it provides for easy validation against an XML Schema, and its xjc Binding Compiler

can be used to easily generate JAXB Java bindings from an existing XML Schema file, greatly facilitating the possible addition of future content formats.

On instantiation, an ECHO10 metadata model object:

1. Instantiates its JAXBElement content tree.
2. Reads the native metadata file, converting contents to XML if needed (e.g. ECS collection metadata converted from ODL format).
3. Unmarshals the native XML into the content tree.
4. Populates its content tree, using values and calculations, and drawing on event, configuration and science data obtained via the Data Model. Code used for calculated metadata was ported from BMGT 8.1 to the extent possible. Detailed documentation on these calculations can be found in the file: BE\_82\_01\_AdditionalMetadataDescription.doc which is enclosed with this review package.
5. Lastly, metadata models also provide a getGeneratedMetadata accessor method, used by the Generator core to obtain the GeneratedMetadata object.

Classes (see class diagrams later in this document):

**Metadata** – Abstract class providing getters for common metadata properties, the GeneratedMetadata accessor, and unmarshalling behavior

**Collection** – Abstract classes providing format-independent Collection behaviors; extends Metadata

**Granule** – Abstract class providing format-independent Granule behaviors; extends Metadata

**Concrete types** – Classes representing metadata with a specific type and format. These classes extend either Collection or Granule, and provide behaviors specific to their particular type and format (e.g. ECSCollection converts its native metadata, the ODL descriptor file, into ECS XML), including methods for calculating metadata and populating their content trees

## **GeneratedMetadata**

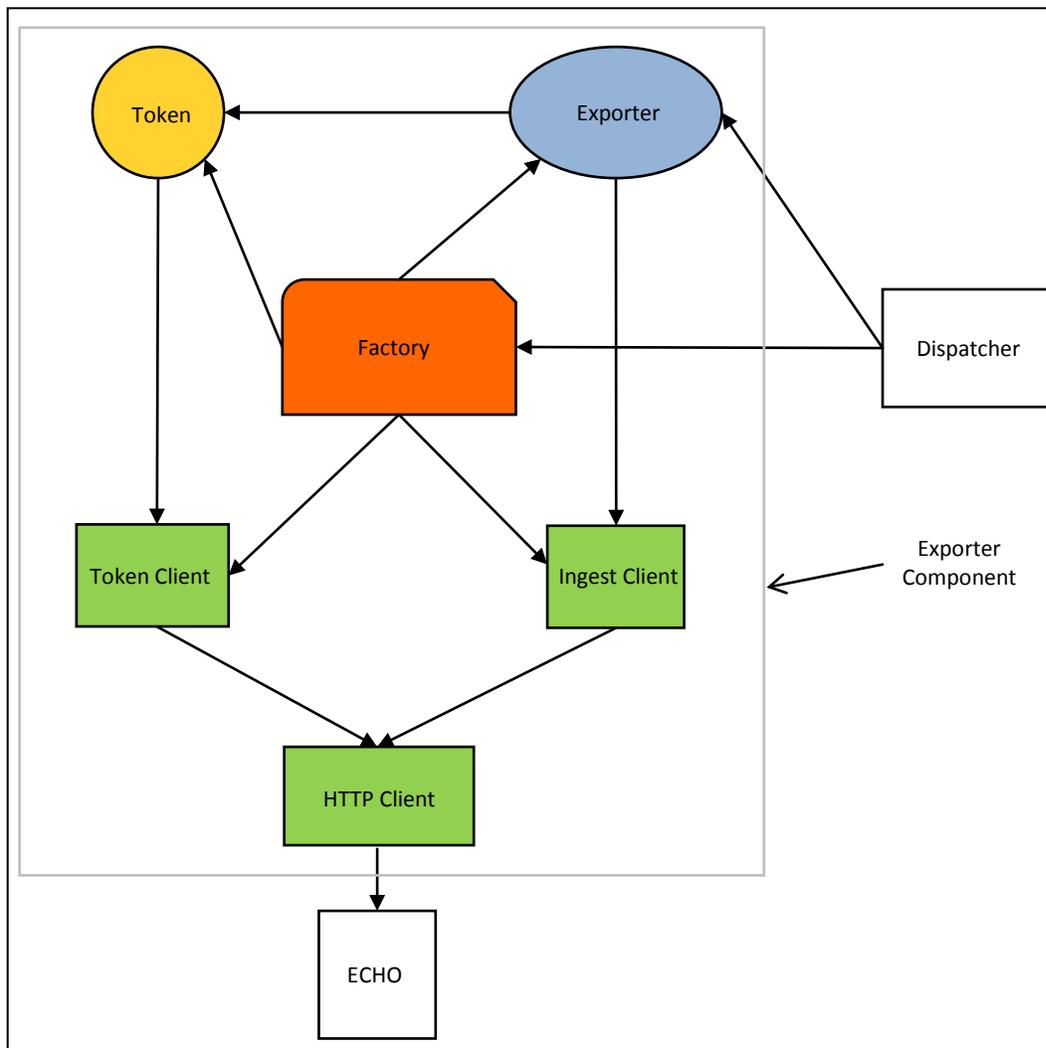
GeneratedMetadata objects encapsulate methods for obtaining the populated, marshaled metadata. This additional layer of encapsulation was chosen (over, say, simply passing around a StreamWriter or String) to allow for a looser, less implementation-specific coupling between Generator and the Data Model and Exporter.

## Additional Metadata Elements

Additional metadata elements which the Generator inserts into the generated XML are described in the document: BE\_82\_01\_AdditionalMetadataDescription.doc which is enclosed with this review package.

### 4.11.6.4 Exporter

Exporter is responsible for sending granule and collection metadata received from Dispatcher to ECHO catalog for ingest. It acts as a client to the ECHO's RESTful Web Services and uses HTTP PUT and DELETE to make the requests. The metadata for export are generated as described in Generator section of the document. The Exporter will return the response received from ECHO to the Dispatcher which in turn passes the response to the Response Handler to be processed.



**Figure 4.11-8. Components of Exporter and Their Dependencies**

## Exporter Sub Components

The figure 4.11-8 shows the major sub-components of Exporter and their Dependencies. These are:

- 1) Token
- 2) Exporter
- 3) HTTP Client and its sub-types
- 4) Factory

Each of the components is described in detail below:

### **Token**

Each RESTful API request to ECHO needs to be authenticated before it is processed. ECHO uses HTTPS for authentication at the protocol level. At the application level, the authentication is done using a security token. Each export request to ECHO needs to have a valid security token in its header before it can be processed. A security token is linked to an ECHO user account, which has been granted ingest privileges via PUMP (Provider User Management Program). ECHO provides a Restful Web Service API, distinct from the API used for ingesting metadata, for creation and management of the Tokens. Tokens are designed as light weight objects within ECHO and many valid tokens can be associated with a single account at any given moment. Each token has a validity period after which it cannot be used for exports. It is also possible to invalidate a valid token using an appropriate request.

The Token component of the Exporter will encapsulate the creation and management of Tokens used for exports to ECHO. Each Exporter object (described next) within a Dispatcher Thread will have its own token which will be used for all the Exports to ECHO by the Exporter object. Hence several valid tokens associated with a single user account will be used concurrently. A token will be created during the creation of Exporter object and will be renewed periodically. The duration that a token will be used will be a configuration property. The token will not be stored in persistent storage- it will be invalidated by a request to ECHO when the BMGT is shut down and a new token will be retrieved when the Exporter is re-initialized. An “orphaned” token caused by a system crash or other fatal system error will automatically be invalidated by ECHO when the token expires and no special action will be taken.

### **Exporter**

This is the primary component involved in the actual export of metadata from ECS to ECHO. An instance of Exporter, also referred to here as an Exporter Object, will live within a single thread inside the Dispatcher and is initialized when a new instance of the thread is created. Several Exporter objects will be running concurrently – one inside each of the multiple threads that run within the Dispatcher. An Exporter object is always associated with a Token and an instance of

HTTP client called Ingest Client (described next). As described earlier, a Token is used for authentication. The HTTP client is used to set the HTTP connection parameters and handle HTTP specific issues like persistence, compression, request retries for certain types of HTTP errors, etc.

An Exporter object accepts an ExportActivity object as the input and requires the ExportActivity to provide an interface to read the metadata contained within it as a string. The string is read into the body of an HTTP request. The Exporter will return the response received from ECHO to the Dispatcher which in turn passes the response to the Response Handler to be processed. But if the Exporter receives an error, depending on the type of error, Exporter will either pass the response back to Dispatcher or try to handle the error itself. If the error is a user authentication error due to invalid token, the Exporter will throw an exception which will cause Dispatcher to pause. But if the error is caused by connectivity or protocol issues, the Ingest Client responds by attempting to resend the request a certain number of times over a short duration of time (several tens of seconds) until it is successful. If, however, all the attempts fail, the error is returned to Dispatcher. The Dispatcher will immediately pause exports and bring the issue to the operator's attention; subsequently it will make several attempts to resume the controller automatically over a longer duration of time (several 10s of minutes) until exports succeeds. But if the Dispatcher resume attempts fail, it will cease auto-resume and resuming Dispatcher will now require operator intervention.

## **HTTP Client and sub-types**

This is the component which directly communicates with the ECHO RESTful interface. There are two different types of HTTP Clients – a) Token Client – This is used by Token to send requests related to management of security tokens. An instance of Token client is transient and is created for every new request to ECHO. That is, an object of this instance is created by the Token for every new request to ECHO that the Token object makes and is discarded after the request is satisfied. Note that a token object might need to speak to ECHO, not just during its creation but also to invalidate the token after it is used for a period of time or before shut down/termination of Exporter object. b) Ingest Client– This is used by the Exporter objects to send the Export requests to ECHO. A single instance of Ingest client is associated with an Exporter object throughout its lifetime in a long running process. It is created during the creation of an Exporter object and is discarded when the Exporter object is discarded.

Various HTTP connection parameters which have direct effect on the performance and consistency of the exports are set here. These parameters will have different value depending on the type of the HTTP Client –Token Client or Ingest Client. Some of the parameters are discussed here:

- a) Protocol – The protocol used for communication with ECHO will be specified here. ECHO currently requires HTTPS. Using HTTPS requires that both the client and the server trust a third party responsible for issuing digital certificates called Certification Authority (CA). EchoHttpClient will depend on the internal java key store (cacerts) to identify the Certification Authorities that it will trust.

- b) Persistence – The Ingest client reuses the HTTP connection from one request to another in accordance with HTTP 1.1 persistence. Persistence could lead to significant improvement in performance. The ingest client will try to reuse the connection as long as the server does not terminate it and it will be released just before terminating the Exporter object. If the server forcefully terminates a connection causing a broken pipe, Ingest client will re-establish the connection. In contrast, the Token client will not reuse the connection. The Token related requests to ECHO are very infrequent and reusing the connection does not lead to significant performance improvements.
- c) Compression - The ECHO web services are capable of processing data transmitted over HTTP connection using the gzip HTTP compression schema to compress the data. HTTP compression capability of a server allows its client to request data from (or post data to) the server in a compressed format if the client so desires. The Ingest client will have a user configurable switch to allow the HTTP compression to be turned on or off on demand by an operator. Depending on the size and the type of the data, this could significantly reduce the network traffic. But, the client and the server are left with additional processing involved in compressing and decompressing the data. Depending on the situation, compression could lead to significant improvement in performance. The compression will be disabled by default. Some experimentation is required to see if turning on the switch leads to better performance. The Token client will not use compression as the size and frequency of requests from Token client are very low and compression would not lead to any significant improvements in performance.
- d) Request Retries –Ingest Client will be configured to retry requests to Exporter if certain kinds of HTTP errors are received. These errors include network issues like “Connection Timeout”, “Unknown Host” or “Connection Interrupted” and protocol errors such as those related to SSL. For these kinds of errors, the client will be configured to retry the request a user configured number of times and over a certain period of time until the request succeeds. If the error persists, the error will be returned to Exporter which in turn passes it to Dispatcher.
- e) Client Identification – The “User-Agent” header of HTTP request will be set to “BMGT Token Client” for Token client and “BMGT Ingest Client” for Ingest Client.
- f) Other parameters – It will be possible to configure other parameters like connection timeout, socket timeout, internal socket buffer used to buffer data while sending/receiving HTTP messages, etc.

## **Factory**

The instances of all the remaining sub-components within Exporter will be created within an instance of Factory. It will have a public interface to create new instances of Exporter Object. The Dispatcher will use an instance of Factory to create Exporter objects which will be used within each of the Dispatcher Threads. An instance of a Factory is created by taking all the

Exporter configuration parameters as input. In order to change the configuration parameters of Exporter, the existing instance of Factory and all the Exporter Objects created using the instance will be discarded and a new Factory object will be created using the changed configuration parameters. New instances of Exporter will now be created using the new Factory Object.

## **Interfaces**

The Exporter will have only one public overloaded function which will be invoked by Dispatcher for export of data to ECHO. The function will accept an ExportActivity object as input and the response received from ECHO is stored in the object itself on return. The method has no return value, but updates the ExportActivity which was passed in.

exportMetadata(exportActivity)

## **Workflow**

- 1) Acquire a new Security Token from ECHO on start-up.
- 2) Setup a new persistent connection or use an existing connection to ECHO for export.
- 3) Set the HTTP method to PUT or DELETE depending on the kind of export. Add the appropriate HTTP header fields.
- 4) Compress the XML content to be exported using gzip compression schema if compression is enabled.
- 5) Send the request to ECHO using the connection.
- 6) If received a network error, attempt to resend the request a pre-configured number of times until successful.
- 7) If an error is received due to expired token, acquire a new token and go back to step 5.
- 8) Decompress the body of the XML response if compression is enabled.
- 9) Return to Dispatcher with the response.
- 10) Terminate the connection to ECHO if no new request for Export is received from Dispatcher with-in a pre-configured duration of time.

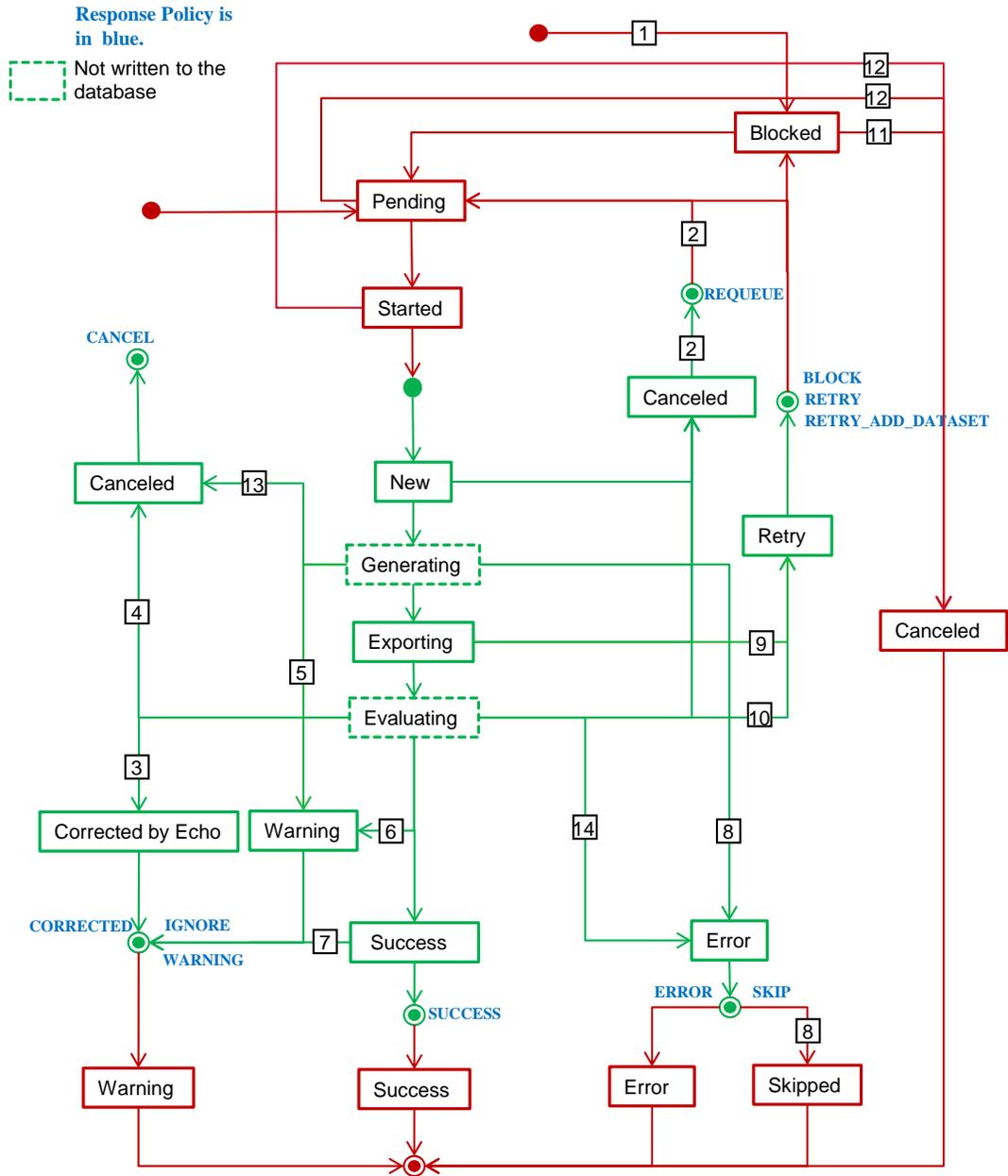
### **4.11.6.5 Response Handler**

#### **Overview**

The Response Handler is responsible for processing the responses returned from ECHO for Export Requests. In addition, Response Handler also handles the cases where an Export Request is skipped, requeued or canceled either due to an error or by an operator action. In the case of a

nominal success response, this involves simply marking the Export Request as complete. But in the case of an error response, the processing could be more complicated.

Figure 4.11-9 shows at a high level the various statuses of Export Request and Export Activity objects and the pathways that exist from the creation of these objects until a terminal status is reached. The figure also shows the “Response policy” which is used in each of the pathways. Response Policy is the action that Response Handler takes either due to a response received from ECHO or due to other special cases like a skipped or canceled request. In the former case, the Response Policy for any given response from ECHO is defined in the table Bg\_Echo\_Error\_Policy.



**Figure 4.11-9. Combined Sequences of Export Request and Export Activity Statuses**

Figure 4.11-9: Diagram showing the combined sequences of Export Request and Export Activity statuses starting from the creation of an Export Request until it reaches a terminal state. The numbered items in the figure are explained below:

1. Occurs for corrective exports of a granule or a collection either due to short form verification errors or errors returned by ECHO for regular exports. In the later case, Response Handler follows `RETRY_ADD_DATASET` response policy.
2. Occurs as a result of Configuration Errors/Fatal Errors which cause Dispatcher to pause or stop without operator intervention. Response Handler follows `REQUE` response policy in this case.
3. Occurs only for verification exports if ECHO returns verification errors. Response Handler follows `CORRECTED` response policy in this case
4. Occurs when Response Handler was supposed to set the status to Retry but the corresponding Export Request was canceled. Response Handler follows `CANCEL` response policy in this case.
5. Occurs if a granule or collection is deleted after an ADD request for it is enqueued. Response Handler follows `IGNORE` response policy in this case.
6. Occurs when a delete request is sent to ECHO but ECHO returns 404 indicating that the item doesn't exist anyway. Response Handler follows `IGNORE` response policy in this case.
7. Occurs when the last export attempt is successful but there exist previous attempts which resulted in Retry. Response Handler follows `WARNING` response policy in this case.
8. Occurs when the Generator indicates that the Export Request should not be sent to ECHO for whatever reason and should be skipped. Response Handler follows `SKIP` response policy in this case.
9. This occurs as a result of certain IO errors. (e.g. `SocketTimeoutException` which occurs if Exporter is able to connect to ECHO, but ECHO did not respond in timely manner.). Response Handler follows `RETRY` response policy in this case.
10. This occurs as result of regular errors (instead of verification errors) returned by ECHO which leads to the export request being either blocked or retried. In this case Response Handler follows one of these response policies depending on the type of error: `RETRY`, `BLOCK` & `RETRY_ADD_DATASET`.
11. Occurs when an Export Request is canceled by the operator after it is blocked.
12. Occurs when an Export Request is canceled by the operator during the Export operation and before the Export Request could reach a terminal state. This is related to 4 and 13.
13. Occurs when the Dispatcher finds that an `ExportRequest` is cancelled after it started processing the Request but before actually sending it to ECHO. Response Handler follows `CANCEL` response policy in this case.

14. This is currently not used anywhere. But could be used if we want to set a response returned from ECHO as an error and not retry the request. Response Handler will follow ERROR response policy in this case.

Each of the Response policies is described below:

- **BLOCK** – If an error returned by ECHO cannot be automatically handled, raise an error which will be displayed to the operator in the GUI and via an email. The Export Request will be placed in a blocked state and can be released by the operator to be re-queued for another attempt, either by invoking a corrective export, or releasing the request via the GUI. The Export Request could optionally be canceled via the GUI. If the number of blocked Export Requests reaches a threshold, Dispatcher may be halted to prevent further errors until the issue can be addressed.
- **RETRY**– If the export is expected to succeed on a retry, then re-queue it (up to a configurable maximum number of times. Depending on configuration, the re-queue may be in a blocked state awaiting operator clearance (for corrective export).
- **RETRY\_ADD\_DATASET** – If the error indicates that the failure was due to another item being missing from ECHO (e.g. granule insert failed because the associated collection is missing). The associated item must be exported before the re-queued original item. Depending on configuration, the re-queue may be in a blocked state awaiting operator clearance (or corrective export).
- **IGNORE** – The error was spurious in nature (e.g. exporting a deletion for an item that was already deleted). The error will be logged, but no additional attention is needed.
- **CORRECTED**– For Long/Incremental Verification. The indicated mismatch was handled by ECHO automatically. The error will be logged as a verification mismatch and available for further investigation, but no immediate attention is needed. Any additional issues incurred during the replacement ingest into ECHO will result in additional errors which will be handled appropriately.
- **WARNING**- If ECHO responds with a success to an Export Request but there exist previous retries which were unsuccessful, Response Handler will mark the Export Request as a warning.
- **SKIP** – If generator finds that an Export Request should not be processed for any reason, it is sent directly to Response Handler from Generator by-passing Exporter. Response Handler will mark the Export Request as being skipped.
- **REQUEUE**- If any of the components within Dispatcher signal a systemic or unknown error, it will cause Dispatcher to either pause or shutdown abnormally, i.e. without operator intervention. In these cases the Export Requests which are being processed when the error occurred will be re-queued.
- **CANCEL** – If an operator issues a cancel request on an Export Request after Dispatcher started processing it, Response Handler can still cancel it if the Request is

not already exported to ECHO by the Exporter. Response handler will mark such a Request as being canceled.

The Response policies described above do not cover all the possible actions that Response Handler takes due to a response received from ECHO. For certain ECHO error responses, which indicate that the error is systemic, Response Handler pauses after sending an alert to the operators. Some of these errors are:

- **Internal Server Error (500)** – When this error is received, operator will be informed that the error indicates an issue which ECHO operations may need to be consulted on. In these cases, the Export Request is re-queued in addition to alerting operator to contact ECHO.
- **Service Unavailable (503)** – This error indicates that ECHO is down, possibly for a scheduled maintenance. In this case Dispatcher will pause after sending an alert to the operator. But Dispatcher will attempt to resume automatically over a period of time defined in a configuration property. If it still fails to export to ECHO, Dispatcher will cease to autoresume and will require operator intervention to resume.
- **Unauthorized (401)** – This indicates that the user is not authorized to perform that transaction. The body of the response will contain an error message. This could happen for example if a token becomes invalid while it is being used. This does not occur in the normal scenario as Tokens are renewed regularly. But if it does occur, Response Handler will send an operator alert and will cause Dispatcher to stop immediately.
- **Access Forbidden (403)** – This indicates the resource is forbidden for accessing. The body of the response will contain an error message. In this case Dispatcher will pause after sending an alert to the operator. But Dispatcher will attempt to resume automatically over a period of time defined in a configuration property.

Table below summarizes all the possible HTTP error codes that ECHO could respond with and the action taken by Response Handler in each case.

Each type of error will be mapped to a particular set of possible outcomes. For instance, an error which indicates that a granule does not exist could either be ignored or require manual intervention, all depending on whether the granule exists in the inventory (if not, then the non existence of the granule in ECHO is correct).

There is one other type of special processing that the Response Handler is responsible for. When a collection is newly enabled for granule export, or has a full update requested, all granules in that collection must be exported. In either of these cases, the Export Request will be marked with a specific export type. When Response Handler sees a successful response for a request of this type, it will automatically queue all of the granules in that collection.

Some DAACs would prefer that automatic ‘corrective’ exports which BMGT determines to be necessary to correct an error not be actually exported until an operator has had a chance to review them, while others are fine with the process being entirely automated except when an error requires manual intervention. For this reason, the Response Handler will be configurable to add

and re queue all events in a ‘blocked’ state requiring an operator to release. Blocked (awaiting operator action) corrective actions will not be much different from the failed requests awaiting manual intervention, but they will be listed separately in the GUI and it will be possible to choose to include only one or the other in a corrective export.

### ECHO HTTP Response codes

Below are the possible HTTP response error codes that ECHO may send back to BMGT, along with what each means, and what BMGT will do in response to each.

HTTP Code	Meaning	Handler Policy
400	Bad Request – This is a general error message saying that something was wrong with the request you sent. It could be invalid XML or an invalid parameter or similar. The body of the response will contain an error message	Alert the operator with the error message, don't pause the exports.
401	Unauthorized – The user you are using is not authorized to perform that transaction. The body of the response will contain an error message	The Response Handler will cause the Dispatcher to halt and alert the operator.
403	Access to the specified resource has been forbidden	The Response Handler will pause the Dispatcher and alert the operator.
404	Not Found – The resource wasn't found. This error could be received if using a bad URL or if retrieving a granule that doesn't exist	Response Handler will ignore this error if this is a delete request.
422	Unprocessable Entity – The entity (granule, database, token, etc) you sent was invalid. This is mostly used during creation or update of something.	Response Handler will handle the response as indicated in the table below. Dispatcher will not be paused.
500	Internal Server Error – Something unexpected happened. ECHO Operations should be notified. It either constitutes a bug in ECHO or the system is misconfigured somehow.	Response Handler will cause the Dispatcher to pause all the Exports to ECHO and contact ECHO and the operator.
503	Service Unavailable – ECHO is down possibly for a scheduled maintenance.	Response Handler will cause Dispatcher to pause, but Dispatcher will attempt to resume automatically over several 10s of minutes. If ECHO is up and running during a reattempt, Dispatcher resumes normal operation, but if auto-resume fails after all the attempts, Dispatcher pauses and requires operator intervention to resume.

## ECHO Error Messages

The table below shows the list of all known item specific error messages that ECHO will return. ECHO does not guarantee that the text of the error messages that it returns will remain constant over time. But each of the error messages is accompanied by a text based code which remains constant even if the message text itself is modified. Each error message maps to a single code though a given code could be mapped to multiple error messages. Response Handler associates a response policy with each of these codes. Next to each item are the possible policies which BMGT will apply in response. Note that the vast majority require manual intervention as a result of blocked requests. This means that the list of error codes mapped explicitly to policies can be quite short.

**Table 4.11-8. ECHO Error Messages for Both Collections and Granules**

HTTP Code	ECHO Error Code	ECHO Message	Response Policy
422	RECORD_INVALID	Campaigns Short names must be unique	BLOCK
422	RECORD_INVALID	Platforms Instruments Characteristics Names must be unique	BLOCK
422	RECORD_INVALID	Online access urls must be unique	BLOCK
422	RECORD_INVALID	Platforms Instruments Short names must be unique	BLOCK
422	RECORD_INVALID	Platforms Instruments Sensors Characteristics Names must be unique	BLOCK
422	RECORD_INVALID	Platforms Instruments Sensors Short names must be unique	BLOCK
422	INVALID_XML	GranuleRecord XML was invalid. Error: Line 4 - cvc-datatype-valid.1.2.1: '20100105T053030.550-05:00' is not a valid value for 'dateTime'.	BLOCK Contact ECHO

**Table 4.11-9. ECHO Error Messages for Both Collections and Granules (1 of 2)**

<b>HTTP Code</b>	<b>ECHO Error Code</b>	<b>ECHO Message</b>	<b>Handler Policy</b>
422	RECORD_INVALID	Additional attributes Names must be unique	BLOCK
422	RECORD_INVALID	Algorithm packages Names must be unique	BLOCK
422	RECORD_INVALID	Associated difs Entry ids must be unique	BLOCK
422	RECORD_INVALID	Collection associations Short name and version ids must be unique	BLOCK
422	RECORD_INVALID	Csdt descriptions Primary csdts must be unique	BLOCK
422	RECORD_INVALID	Platforms Characteristics Names must be unique	BLOCK
422	RECORD_INVALID	Platforms Short names must be unique	BLOCK
422	RECORD_INVALID	Two d coordinate systems Names must be unique	BLOCK
422	RECORD_INVALID	Two d coordinate systems Coordinate 1 Minimum must be less than the maximum	BLOCK
422	RECORD_INVALID	Collection additional attribute [alpha] was of Data Type? [INT], cannot be changed to [FLOAT]	BLOCK
422	RECORD_INVALID	Collection additional attribute [alpha] is referenced by existing granules, cannot be removed.	BLOCK
422	RECORD_INVALID	Collection campaign [alpha] is referenced by existing granules, cannot be removed.	BLOCK
422	RECORD_INVALID	Collection end_date_time [2000-07-01T12:00:00Z] must be later than existing Granule end_date_time [2000-08-01T12:00:00Z]	BLOCK

**Table 4.11-9. ECHO Error Messages for Both Collections and Granules (2 of 2)**

HTTP Code	ECHO Error Code	ECHO Message	Handler Policy
422	LONG_NAME_VERSION_NOT_UNIQUE	LongName [xxxx] and VersionId [xx] is already defined by existing DatasetId [A minimal valid collection V 1]	BLOCK
422	SHORT_NAME_VERSION_NOT_UNIQUE	ShortName [xxxx] and VersionId [xx] is already defined by existing DatasetId? [A minimal valid collection V 1]	BLOCK
422	DATASET_ID_NOT_MATCH	DatasetId [foo] does not match what is defined in the xml [A minimal valid collection V 1]	BLOCK OR IGNORE? Mismatch Handled by ECHO
422	RECORD_INVALID	LastUpdate [2002-01-01T12:00:00Z] is earlier than LastUpdate [2002-01-01T12:00:01Z] of the existing DatasetRecord	BLOCK Contact ECHO
422	RECORD_INVALID	Geometry Bounding rectangle 1 North should be greater than south	BLOCK
404	REST_ITEM_NOT_FOUND	Unable to find dataset with dataset_id XXXXX	IGNORE
422	NOT_ALL_GRANS_INDEXED	Not all granules of the collection are indexed when updated request for the collection is sent.	RETRY

**Table 4.11-10. ECHO Error Messages for Granules Only (1 of 3)**

HTTP Code	ECHO Error Code	ECHO Message	Handler Policy
422	RECORD_INVALID	Additional attributes name [Attribute Name-X] must reference an additional attribute in the dataset	BLOCK
422	RECORD_INVALID	Additional attributes Values 'pi' is not a valid float	BLOCK

**Table 4.11-10. ECHO Error Messages for Granules Only (2 of 3)**

HTTP Code	ECHO Error Code	ECHO Message	Handler Policy
422	RECORD_INVALID	Campaigns short name [Short Name-250] must reference a campaign in the dataset	BLOCK
422	DATASET_NOT_DEFINED	Dataset with DatasetId [XXX] is not defined	RETRY_ADD_DATASET
422	DATASET_NOT_DEFINED	Dataset with ShortName [XXX] and VersionId [XXX] is not defined	RETRY_ADD_DATASET
422	RECORD_INVALID	Platforms Instruments Characteristics name [XXX] must reference a characteristic in the dataset	BLOCK
422	RECORD_INVALID	Platforms Instruments short name [Short Name-XX] must reference an instrument in the dataset	BLOCK
422	RECORD_INVALID	Measured parameters Parameter names must be unique	BLOCK
422	RECORD_INVALID	Platforms Instruments Operation modes must be unique	BLOCK
422	RECORD_INVALID	LastUpdate [2002-01-01T12:00:00Z] is earlier than LastUpdate [2002-01-01T12:00:01Z] of the existing GranuleRecord	BLOCK <b>Contact ECHO</b>
422	RECORD_INVALID	Platforms Instruments short name [XXX] must reference an instrument in the dataset	BLOCK
422	RECORD_INVALID	Platforms Instruments Characteristics name [XXX] must reference a characteristic in the dataset	BLOCK
422	RECORD_INVALID	Platforms Instruments Sensors short name [XXX] must reference a sensor in the dataset	BLOCK
422	RECORD_INVALID	The granule BeginningDateTime is earlier than Dataset BeginningDateTime	BLOCK
422	RECORD_INVALID	The granule EndingDateTime is later than Dataset EndingDateTime	BLOCK

**Table 4.11-10. ECHO Error Messages for Granules Only (3 of 3)**

HTTP Code	ECHO Error Code	ECHO Message	Handler Policy
422	RECORD_INVALID	The granule SingleDateTime is earlier than Dataset BeginningDateTime	BLOCK
422	RECORD_INVALID	The granule SingleDateTime is later than Dataset EndingDateTime	BLOCK
422	RECORD_INVALID	Two d coordinate system Start coordinate 1 [-1.0] is less than the minimum [0.0] defined in the dataset	BLOCK
422	RECORD_INVALID	Two d coordinate system Start coordinate 1 [11.1] is greater than the maximum [11.0] defined in the dataset	BLOCK
422	RECORD_INVALID	Two d coordinate system name [not_defined] must reference a two d coordinate system in the dataset	BLOCK
422	RECORD_INVALID	Geometry Gpolygon 1 has the following Oracle spatial validation error: ORA-13373: invalid line segment in geodetic data [Element <1>] [Ring <1>]	BLOCK
422	GRANULE_UR_NOT_MATCH	GranuleUR [foo] does not match what is defined in the xml [GranuleUR100]	BLOCK OR IGNORE?! <b>Mismatch Handled by ECHO</b>
422	RECORD_INVALID	Geometry must be provided when the parent collection's GranuleSpatialRepresentation is GEODETIC	BLOCK
404	REST_ITEM_NOT_FOUND	Unable to find granule with granule_ur XXXXX	IGNORE

## Architecture

The Response Handler will be a fairly simple component. It will query the database for a list of errors which are not handled by the 'Manual Intervention' policy, and will instantiate a hash table linking error messages to policies. When the Response Handler is invoked, it will look up the error in its table, if it does not see the error, it will send an email to the operator and block the

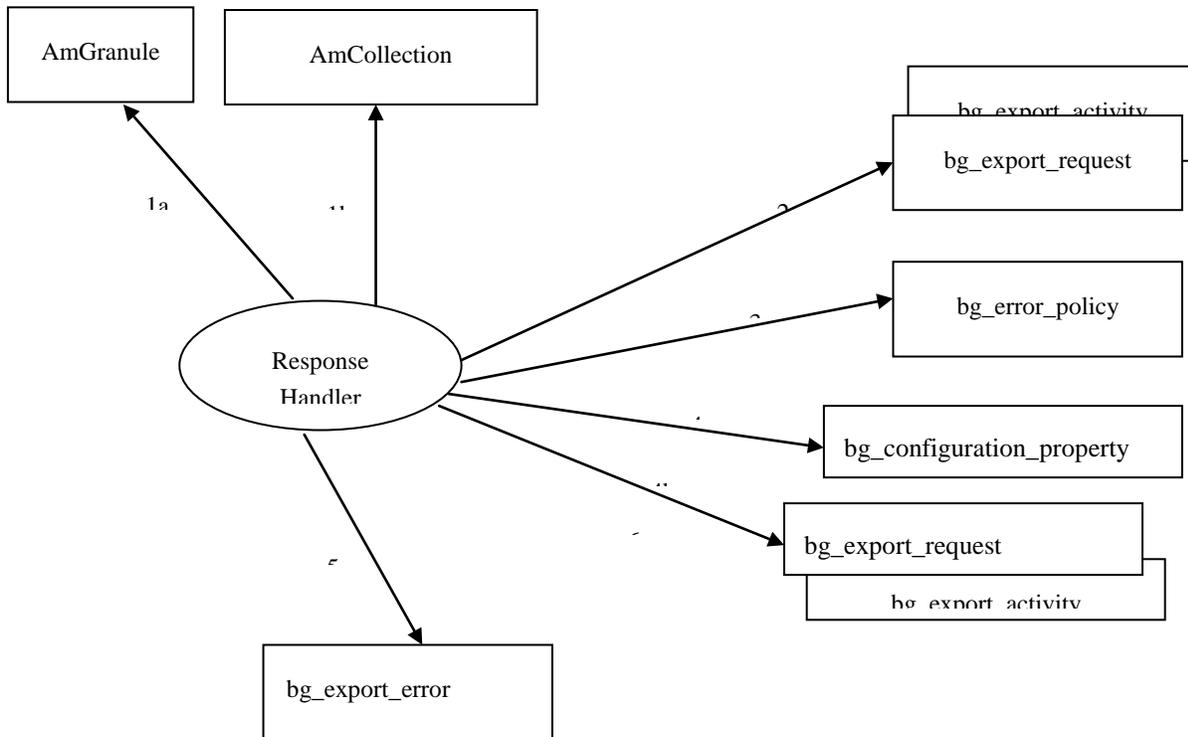
request (Manual Intervention) and otherwise will invoke the appropriate policy. Each policy will be implemented as a class, all of which implement the same interface.

## Interfaces

The Response Handler will expose 3 public methods - first one takes in an ExportActivity object and other two take in an Exception in addition to ExportActivity object. The exceptions correspond to unusual cases in which Export Activity needs to be set to a terminal state even before exporting to ECHO such as skipped or canceled requests. In one case, the Exception is caused by request specific errors and in the other case it is caused by systemic errors which lead to paused Dispatcher.

## Workflow

A high level view of the Response Handler's interaction with the database is illustrated below.



**Figure 4.11-10. Manual Export Process Database Sequence**

1. Get status of the referenced inventory item
  - a. Granule.
  - b. Collection.
2. Get record for the responsible export request and activity
3. Get error handler for the specified error
4. Handle Error
  - c. Get notification email address
  - d. Add new request to queue
5. Update error record
6. Update export request (to re-queue, mark as complete, or mark as blocked awaiting manual intervention).
  - e. May require adding row to export request or activity table.

#### **4.11.6.6 Catalog Event (Automatic) Driver**

The Catalog Event (or Automatic) Driver is the component responsible for driving the export of metadata in response to normal system events. The Driver transforms events in the AIM Events table into requests in the BMGT Export Request Queue. This driver runs as a daemon constantly polling for new events to move over on a frequent interval. At each polling interval, the Driver will select any new events from the AIM event queue.

As part of the selection of events, the Driver will ensure that only events which are eligible for export are added to the queue. For instance, all events associated with a collection which is not enabled for export will be ignored (but marked as having been analyzed and not available for later pickup), and granule events associated with a collection which is enabled for collection metadata export only will be ignored.

The Driver can be suspended by the operator if for any reason it is necessary to halt the enqueuing of new export entries. This suspension (and the subsequent resumption) can be performed via the BMGT GUI.

The ideal polling interval for the driver will depend on DAAC preferences, 0 minutes or so should be sufficiently frequent to ensure prompt export of updates, but infrequent enough to reduce redundant requests being enqueued for the same granule or collection.

#### **4.11.6.7 Manual Driver**

The Manual Driver allows the operator to manually initiate the export of metadata to ECHO for any catalog items they desire. Manual Export is used in instances where the normal, automatic export of metadata based on catalog updates is not sufficient. For example, if granules have been updated in a way that has not triggered an update event in the AIM Events table, manual export could be used to ensure those changes are reflected in ECHO. In short, Manual export allows the operator to export whatever metadata they deem to be necessary in a simple and easy manner.

The Manual Driver accepts as input the following parameters:

- **Metadata type:** Granule, Collection, or both.
- **Item list:** List of Granule ids, Collection ShortName/VersionIds, and/or Collection Group names, either on the command line or in a referenced file. These may all be in the same command line option, with the driver smart enough to figure out what type of item is being referenced.
- **Time range:** Start and/or end datetime used to select granules within a collection. Also, an indicator of whether the datetime selection should apply to insert or last update time.
- **Scripting options:** whether to override any interactive prompts and/or retry on error or blockage(due to a concurrently running driver)
- **Export Type:** Special options to specify the desired type of export (e.g. insert-only or delete-only). Also, allows the operator to request a full collection update (delete collection and all its granules, then re export).

The Driver will be instantiated either from the GUI or from the command line. When it is run, the input parameters will be used to populate the BMGT Export Request Queue with the appropriate records. These requests will then be picked up by the BMGT Dispatcher, and the appropriate metadata will be exported in the same fashion as with an automatic export. Requests will be inserted and processed in such a way that the appropriate action is taken in ECHO. For instance, any valid granule which is listed in operator input will result in a granule insert being exported, but any granule ID which does not exist in the database or is logically deleted will be exported as a delete. This is all transparent to the operator – basically, the operator tells BMGT to export the current state of the specified items. The operator can however specify that they would only like to enqueue items which would result in insert, or those which would result in delete (insert-only and delete-only options).

There is one special case and exception to the above. There are cases where a modification to Collection metadata will require a complete removal of the collection and its associated granules from ECHO. In these cases, the manual Driver must allow an operator to request the export of a deletion of a non deleted Collection, as well as the re export of the new collection metadata and all granules in the collection. These situations will be marked in the export request such that the generator will export the deletion, and then the response handler will enqueue the collection re insert once the deletion has succeeded. Once the collection re insert has succeeded, the Response Handler will enqueue the export of all granules in that collection.

The manual driver can also be run in ‘corrective’ mode, where it will enable and re enqueue any ‘blocked’ requests, filtered by the operator’s options (e.g. only for a particular collection). ‘blocked’ requests are those which have failed and need operator analysis before being retried. Also, depending on the configuration, new requests enqueued in response to errors received from ECHO may be enqueued in the ‘blocked’ state.

#### 4.11.6.8 Verification Driver

The Verification Driver is responsible for adding events to the BMGT request queue for the purpose of verification exports. Verification exports are used to periodically and/or systematically verify the ECS holdings against the ECHO catalog.

The Verification Driver is essentially an extension of the Manual Driver and as such, most of the input options described in the Manual Driver section also apply to Verification (at least in Long and Incremental modes). As in previous releases, the verification and manual drivers will be the same software entity, but due to their different use cases and options, they are documented separately.

There are three types of verification export, Long Form, Incremental, and Short Form.

**Long Form Verification** exports the full metadata of selected items to ECHO for comparison against their current holdings. A Long Form verification would generally be performed manually when the DAAC staff has reason to suspect a discrepancy between DAAC holdings and the ECHO catalog (due to a bug or other error). However, Long Form verifications could also be done on a schedule and kicked off by a cron. To initiate a Long Form Verification, an operator specifies the items to include (by granule or collection identifier) and the type of metadata to verify (Granule or Collection). The operator can also optionally specify an insert or lastUpdate time range to further filter granules in any selected collections. Once the items are selected, the Verification Driver verifies that all specified items are valid and are enabled for export. The Driver then adds corresponding rows to the BMGT Export Request Queue for each item. These rows are similar to normal export requests, but are flagged as verification requests. The requests are then worked off by the Dispatcher. The only difference from normal metadata exports is that a special HTTP header will be placed in each export by the Exporter which will flag the export as a verification.

When ECHO receives an export with the Verification HTTP header, instead of simply ingesting the metadata, it will first check to see if the item already exists in its catalog, and if so, do a comparison of all elements in the metadata. If the item does not exist, or if there is a discrepancy found, the new metadata will be ingested, replacing any current metadata for that item, and a warning notice will be returned to BMGT indicating the nature of the discrepancy. It is also possible that the ingest attempt results in an error as well, in which case additional ingest errors will be returned. BMGT will process the errors using the normal Response Handler mechanism, resulting, potentially, in events being queued for re export of catalog items. If the response included only warnings, indicating that there was a discrepancy, but it was repaired by the ingest attempt, then these will be logged and displayed to the operator in the GUI, but will not cause any additional actions to be taken.

**Incremental Verification** is a sub-type of Long Form Verification. The only difference is in how items are selected for inclusion in the export. Instead of the operator explicitly selecting collections and/or granules to be verified, for an Incremental Verification the Verification Driver automatically determines what items are eligible for verification. The intention of Incremental Verification is to start at the granule in the system which has been unmodified for the longest time (earliest LastUpdateTime), and then move forward from that granule, verifying each until

the most recently updated granule is reached. Once the Verification has caught up to the most recent granule, then it will continue, verifying newly updated granules shortly after they are exported. The system can also be reset to start once again at the first granule (either for the entire system or a particular collection). There is a configured maximum number of granules per verification set and also a maximum number of granules per collection per set to ensure that the verification export is kept to a reasonable volume and does not drown out the normal, automatic export of new events. The Verification Export can be kicked off either manually, or via a cron.

Once the items for an Incremental Verification are selected, the associated requests are added to the BMGT Export Request Queue, and the process continues in the same manner as for Long Form Verification.

**Short Form Verification** is very different from the other verification types. Rather than verifying the full metadata of each item, Short Form Verification performs a simple existence check between the ECS and ECHO holdings. These verifications will also include Last Update Time to help determine with a coarse granularity whether any updates have failed to be ingested in ECHO. Last update time comparisons will, however, allow for a certain amount of difference before flagging an error in order to account for any rounding or truncation. Also unlike the other forms of Verification, Short Form is done via a query to ECHO rather than an export. To perform a Short Form Verification, the operator would select the type of metadata (Granule or Collection) to verify. For Collection metadata, a query will be made to ECHO for a list of all collections, and this list will be compared against the list of collections configured for BMGT export. For Granule metadata, the operator must specify additionally one or more collections. The Driver will then query ECHO for all of the granules in the specified collection(s) and compare it against the list in the AIM database. The size of the list of granules in a collection could easily be in the millions, so checks will be done to avoid verifying too many collections at once. In order to work however, an entire collection must be verified at once.

Once the listing retrieved from ECHO is compared against the list in the AIM database, any discrepancies will be noted and added to the BMGT Export Request Queue, optionally in a blocked state, awaiting operator approval before being exported, as per DAAC configuration. These requests will then be processed as any other export request via the Dispatcher.

#### **4.11.6.9 BMGT CSCI Context**

BMGT consists of only one CSCI. Therefore, the Subsystem Context in Figure 4.11-1 represents the BMGT CSCI Context, and it will not be replicated here.

#### **4.11.6.10 BMGT CSCI Process Interface Description**

BMGT consists of only one CSCI. Therefore, the interface description in Table 4.11-1 represents the BMGT CSCI Process Interface, and it will not be replicated here.

#### **4.11.6.11 Data Stores**

BMGT uses AIM database as well as the StorNext XML archive to generate its products. Table 4.11-6 describes the Data Stores.

**Table 4.11-11. Data Store**

Data Store	Type	Description
Inventory DB	Postgres	BMGT reads required Metadata info from Inventory DB.
Small file archive	StorNext file system	BMGT reads granule and collection metadata files from the XML archive, applies XSLT stylesheets to them, and exports the resulting XML.

#### 4.11.6.12 BMGT GUI Functional Overview

The BMGT GUI is a JavaScript/Dojo based web GUI which will allow the operator to monitor the generation and export of BMGT requests (Automatic, Manual, Corrective and Verification).

The GUI provides DAAC staff with the following functions:

- Display BMGT export processes that are currently queued or in progress
- Allow the operator to pause / resume exports to ECHO
- List the most recent export requests and view detailed information about them
- Cancel an export request
- List the most recently completed exports which resulted in errors and view detail information about them.
- View and change some BMGT configuration parameters. Changing the BMGT configuration parameters will be restricted to DAAC staff that is logged in as BMGT administrator
- Display global alerts upon a configured number of failures
- Display the status of incremental verification on a system, group, and collection level and reset the incremental verification of a particular collection.
- Display and remove alerts flagged by repeatedly failing BMGT exports.

#### 4.11.6.13 ECHO Metadata Schemas

The ECHO schemas are very large and would not conveniently fit in to this document. The latest schema currently is use in the operational ECHO environment can be found at the URLs listed below.

##### 4.11.6.13.1 Collection.xsd

See <http://www.echo.nasa.gov/ingest/schemas/operations/Collection.xsd>

##### 4.11.6.13.2 Granule.xsd

See <http://www.echo.nasa.gov/ingest/schemas/operations/Granule.xsd>

#### **4.11.6.13.3 Browse.xsd**

See <http://www.echo.nasa.gov/ingest/schemas/operations/Browse.xsd>

#### **4.11.6.13.4 MetadataCommon.xsd**

See <http://www.echo.nasa.gov/ingest/schemas/operations/MetadataCommon.xsd>

# Abbreviations and Acronyms

---

## A

ABC++	Document Generator used to provide class level detail
ACL	Access Lists
ACMHW	Access and Control Management Hardware (Configuration Item)
AD	Advertisement
ADC	Affiliated Data Center (National Oceanic and Atmospheric Administration only)
AGS	ASTER Ground System
AIT	Algorithm Integration and Test
AIM	Archive Inventory Management
AITHW	Algorithm Integration and Test Hardware (Configuration Item)
AI&T	Algorithm Integration and Test
AITTL	Algorithm Integration and Test Tools (Computer Software Configuration Item)
ALOG	Applications Log
AM-1	See TERRA (spacecraft)
AOI	Area of Interest
AOS	ASTER Operations Segment
AP	Algorithm Package
APC	Access/Process Coordinators
API	Application Program Interface
AQA	Algorithm Quality Assurance
AQUA	PM-1 Satellite (AIRS, AMSR-E, AMSU, CERES, HSB, MODIS)
AR	Action Request
AS	Administration Stations
ASCII	American Standard Code for Information Interchange
ASE	Adaptive Server Enterprise

ASTER Advanced Spaceborne Thermal Emission and Reflection Radiometer  
ATM Asynchronous Transfer Mode  
AURA NASA mission to study the earth's ozone, air quality and climate (formerly the CHEM mission)

## **B**

BBR Browse metadata (conforming to the ECS/ECHO Metadata Inventory ICD)  
BCP Bulk Copy Program  
Bulk Copy Procedure  
BDS Bulk Data Server  
BMGT ECS Bulk Metadata Generator Tool  
BLM Baseline Manager

## **C**

CAD Computer Aided Design  
CCB Change Control Board (Raytheon Convention)  
Configuration Control Board (NASA Convention)  
CBLM ClearCase Baseline Manager  
CCDI ClearCase DDTS Integration  
CCR Configuration Change Request  
CDE Common Desktop Environment  
CDR Critical Design Review  
CDRL Contract Data Requirements List  
CD-ROM Compact Disk - Read Only Memory  
CDS Cell Directory Service  
CFG Configuration File  
CGI Common Gateway Interface  
CHUI Character-based User Interface  
CI Configuration Item  
CLI Command Line Interface  
CLS Client Subsystem  
CM Configuration Management

CMI	Cryptographic Management Interface
CMP	Configuration Management Plan
CN	Change Notice
CO	Contracting Officer
COTS	Commercial Off the Shelf (Software or Hardware)
CPF	Calibration Parameter File
CPU	Central Processing Unit
CRM	Change Request Manager
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSMS	Communications and Systems Management Segment (ECS)
CSS	Communications Subsystem
<b><u>D</u></b>	
DAAC	Distributed Active Archive Center
DADS	Data Archive and Distribution System
DAO	Data Assimilation Office
DAP	Delivered Algorithm Package
DAS	Dual Attached Station
DB	Database
DBMS	Database Management System
DCCI	Distributed Computing Configuration Item
DCN	Document Change Notice
DDICT	Data Dictionary (Computer Software Configuration Item)
DDR	Detailed Design Review
	Data Delivery Record (same as a Product Delivery Record)
DDT	DAAC Distribution Technician
DDTS	Distributed Defect Tracking System (COTS)
DEM	Digital Elevation Model
DESKT	Desktop (Computer Software Configuration Item)
DEV	Custom Developed Code

DFS	Distributed File System
DID	Data Item Description
DIPHW	Distribution and Ingest Peripheral Hardware Configuration Item
DLL	Dynamic Link Library
DLT	Digital Linear Tape
DM	Data Management
DMGHW	Data Management Hardware (Configuration Item)
DMS	Data Management Subsystem
DNS	Domain Name Service
DOF	Distributed Object Framework
DORRAN	Distributed Ordering, Researching, Reporting, and Accounting Network (At EDC)
DP	Data Provider
DPAD	DataPool Action Driver
DPL	Data Pool Subsystem
DPR	Data Processing Request
DPRID	Data Processing Request Identifier
DPREP	Data Pre-Processing
DR	Data Repository
DRPHW	Data Repository Hardware (Configuration Item)
DSC	Development Solution for the C programming language
DSS	Data Server Subsystem
DTD	Document Type Definition
DTF	Sony DTF Tape cartridge system (replacement for the D3 tape cartridge system)
DTS	Distributed Time Service
<b><u>E</u></b>	
EBIS	EMD Baseline Information System
ECHO	ECS ClearingHouse
ECN	Engineering Change Notice
ECS	Earth Observing System Data and Information Core System
EDC	Earth Resource Observation System (EROS) Data Center

EDF	ECS Development Facility
EDG	EOS Data Gateway
EDHS	ECS Data Handling System
EDN	Expedited Data Set Notification
EDOS	Earth Observing System Data and Operations System
EDR	Expedited Data Set Request
EDS	Expedited Data Set
EC	Error conditions (in tickets)
METG/C	Granule/Collection metadata
EED	EOSDIS Evaluation and Development
EGS	EOS Ground System
EISA	Enhanced Industry Standard Architecture
E-mail	Electronic Mail (also Email, e-mail, and email)
EMD	EOSDIS Maintenance and Development Project
EMOS	ECS Mission Operations Segment (formerly FOS)
EMSn	EOSDIS Mission Support network
EOC	Earth Observing System Operations Center
EOS	Earth Observing System
EOSDIS	Earth Observing System Data and Information System
EPD	External Product Dispatcher
EROS	Earth Resource Observation System
ESDIS	Earth Science Data and Information System (GSFC Code 505)
ESDT	Earth Science Data Type
ESRI	Environmental Systems Research Institute
ETM+	Enhanced Thematic Mapper Plus (Landsat 7)
EWOC	ECHO WSDL Order Component
<b><u>F</u></b>	
FC	Functional components (capabilities in tickets)
FCAPS	Fault, Configuration, Accountability, Performance, and Security services
FDS	Flight Dynamics System

FH	Fault Handling
FLDB	Fileset Location Database
F&PRS	Functional and Performance Requirements Specification
FSMS	File and Storage Management System
FTP	File Transfer Protocol
FTPD	File Transfer Protocol Daemon
<b><u>G</u></b>	
GB	gigabyte ( $10^9$ )
Gb	gigabit ( $10^9$ )
GCDIS	Global Change Data and Information System
GCMD	Global Change Master Directory (not developed by ECS)
GFE	Government Furnished Equipment
GLAS	Geoscience Laser Altimeter System
GSFC	GODDARD Space Flight Center (NASA facility and DAAC)
GSMS	Ground System Management Subsystem (ASTER)
GTWAY	(ASTER) Gateway (Computer Software Configuration Item)
GUI	Graphical User Interface
<b><u>H</u></b>	
HDF	Hierarchical Data Format
HDF-EOS	an EOS proposed standard for a specialized HDF data format
HEG	HDF-EOS To GeoTIFF
HMI	Human Machine Interface
HSB	Humidity Sounder for Brazil
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
HW	Hardware
HWCI	Hardware Configuration Item
<b><u>I</u></b>	
IAS	Image Assessment System

IBM	International Business Management
ICESat	Ice, Cloud and Land Elevation Satellite
ICD	Interface Control Document
ID	User Identification (or Identifier)
IDG	Infrastructure Development Group
IDL	Interactive Data Language
I/F	Interface
IGS	International Ground Station (Landsat 7)
IHCI	Internetworking hardware configuration item
IIU	DSS Inventory Insert Utility
ILG	Infrastructure Library Group
ILM	Inventory, Logistics, Maintenance (ILM) Manager
IMS	Information Management System (ECS element name)
INHCI	Internetworking HWCI
I/O	Input/Output
IOS	Internetwork Operating System
IP	InterProcess Communication
IPC	Inter-Process Communication
IRD	Interface Requirements Document
IRR	Incremental Release Review
ISIPS	ICESat Science Investigator-Led Processing System
ISO	International Standards Organization
ISR	ECHO Ingest Summary Report
ISS	Internetworking Subsystem
I&T	Integration and Test
<b><u>J</u></b>	
JAF	Java Activation Framework
JAVA	Programming Language
JAXP	JAVA APL for XML Processing
JDBC	Java Database Connectivity

JDOM	Java Document Object Model
JESS	Java Earth Science Server
JEST	Java Earth Science Tool
JPL	Jet Propulsion Laboratory (DAAC)
JRE	Java Runtime Environment
<b><u>K</u></b>	
KFTP	Kerberos File Transfer Protocol
<b><u>L</u></b>	
L0 - L4	Level-0 through Level-4 data (ECS)
L0R	Landsat Reformatted Data
LAMS	Landsat 7 Archive Management System
LAN	Local Area Network
LaRC	Langley Research Center (DAAC)
LFS	Local File System
LZ77	Lempel-Ziv coding
<b><u>M</u></b>	
M&O	Maintenance and Operations
MB	Megabyte (10 <sup>6</sup> )
Mbps	Megabits Per Second
MCF	Metadata Configuration File/Metadata Control Files
MCI	Management Software Configuration Item (Computer Software Configuration Item)
METC	Collection Metadata (conforming to the ECS/ECHO Metadata Inventory ICD), also ECSMETC
METG	Granule Metadata (conforming to the ECS/ECHO Metadata Inventory ICD), also ECSMETG
METU	Update Metadata (conforming to the ECS/ECHO Metadata Inventory ICD), also ECSMETU
MISR	Multi-Imaging SpectroRadiometer
MLCI	Management Logistics Configuration Item (Computer Software

	Configuration Item)
MM	Mode Management
MMO	Mission Management Office
MMS	Mode Management Service
MOC	Mission Operations Center
MOPITT	Measurements of Pollution in the Troposphere
MP	Message Passing
MS	Mass Storage
MSCD	Mirror Scan Correction Data (file)
MSS	System Management Subsystem
MSSHW	(System) Management (Subsystem) Hardware Configuration Item
MTA	LAMS Metadata File
MTP	Distribution Product Metadata File Extension (<filename>.MTP)
MTPE	Mission to Planet Earth
<b><u>N</u></b>	
NASA	National Aeronautics and Space Administration
NBSRV	Spatial Subscription Server
NCEP	National Centers for Environmental Predictions
NCR	Non-conformance Report
NESDIS	National Environmental Satellite, Data, and Information Service (NOAA)
NFS	Network File System
NIS	Network Information Service
NISN	NASA Integrated Services Network
NMC	National Meteorological Center (located at National Oceanic and Atmospheric Administration - NOAA)
NNTP	Network News Transfer Protocol
NOAA	National Oceanic and Atmospheric Administration
NSI	National Aeronautics and Space Administration Science Internet
NSIDC	National Snow and Ice Data Center (DAAC)

## **Q**

ODL	Object Description Language
OEA	OGC-ECHO Adaptor
OEM	Original Equipment Manufacturer
OMS	Order Manager Subsystem
OMSHW	Order Manager Subsystem Hardware
OMSRV	Order Manager Server
OMGUI	Order Manager Graphical User Interface
OOA	Object oriented analysis
OOD	Object oriented design
OODCE	Object Oriented distributed computing environment
OPS CON	Operations Concept
OS	Operating System
OSI	Open Systems Interconnect

## **P**

PAN	Production Acceptance Notification
PC	Personal Computer
	Performance Constraints (in tickets)
PCD	Payload Correction Data (file)
PCFG	Process Configuration File
PDR	Product Delivery Record
PDRD	Product Delivery Record Discrepancy
PF	Process Framework
PI	Principal Investigator
PM-1	EOS Afternoon Equator Crossing Mission (See Aqua); Mission to study the land, oceans and the earth's radiation budget
PMPDR	Physical Media Product Delivery Record
PSA	Product Specific Attributes
PVC	Performance Verification Center

## **Q**

QA	Quality Assurance
QDS	Quick Look Data Set (Same as Expedited Data Set)
QAUU	DSS Quality Assurance Update Utility

## **R**

RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
RCS	Request Communications Support
RDBMS	Relational Database Management System
REL	Release
RFA	Remote File Access
RFC	Request For Comments
RIP	Routing Information Protocol
RMA	Reliability, Maintainability and Availability
RMS	Request Management Services
ROSE	Request Oriented Scheduling Engine
RPC	Remote Procedure Call
RSC	Raytheon Systems Company
RTU	Rights to Use

## **S**

S4PM	Simple, Scalable, Script-based Science Processor for Missions
SAGE III	Stratospheric Aerosol and Gas Experiment III
SAN	Storage Area Network
SAS	Single Attached Station
SATAN	Security Administrator Tool for Analyzing Networks
SCF	Science Computing Facility
SCLI	SDSRV Command Line Interface
SCP	Secure Copy
SCSI	Small Computer System Interface

SDE	Software Development Environment
SDF	Software Development Folder
SDP	Science Data Processing
SDPTK	Science Data Processing Toolkit Science Data Processing Toolkit (Computer Software Configuration Item)
SDSRV	Science Data SeRVer (Computer Software Configuration Item)
SGI	Silicon Graphics, Inc.
SIM	Spectral Irradiance Monitor
SIPS	Science Investigator-Led Processing Systems
SMC	System Management Center System Monitoring and Coordination Center
SMP	Symmetric Multi-processor
SMTP	Simple Mail Transport Protocol
SNAC	StorNext Archive Cash
SOAP	Simple Object Access Protocol
SOLSTICE	Solar Stellar Irradiance Comparison Experiment
SORCE	Solar Radiation and Climate Experiment
SPARC	Single Processor Architecture
SPRHW	Science Processing Hardware (Configuration Item)
SQL	Structured Query Language
SQS	Spatial Query Server
SRF	Server Request Framework
SSAP	Science Software Archive Package
SSH	Secure Shell
SSIT	Science Software Integration and Test
SSS	Spatial Subscription Server Subsystem
STK	StorageTek
SW	Software
Sybase	(ECS) COTS database management product (ASE)
SYSLOG	System Log

## **T**

TAR	Tape Archive
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TD	Technical Documents
TELNET	Telecommunications Network
TERRA	EOS AM Mission spacecraft 1, morning equator crossing spacecraft series -- ASTER, MISR, MODIS and MOPITT instruments; Mission to study the land, oceans and the earth's radiation budget
TIM	Total Irradiance Monitor
TM	Thematic Mapper (Landsat)
TT	Trouble Ticket
TTPro	TestTrack Pro

## **U**

UDP	User Datagram Protocol
UML	Unified Modeling Language
UR	Universal Reference
URL	Universal Resource Locator
USGS	U. S. Geological Survey
UUID	Universal Unique Identifier

## **V**

VT	Virtual terminal
----	------------------

## **W**

WAN	Wide Area Network
WIST	Warehouse Inventory Search Tool
WKBCH	WorKBenCH (Computer Software Configuration Item)
WKSHW	Working Storage Hardware Configuration Item
WRS	Worldwide Reference System
WS	Working Storage
WSDL	Web Service Definition Language

WWW

World Wide Web

**X**

xAR

x Acquisition Request (where x is any kind of or generic acquisition request)

XBDS

Bulk Data Service Protocol

XDR

External Data Representation

XFS

Extended File System

XML

eXtensible Markup Language

XSD

XML Schema

XRU

DSS XML Replacement utility

XSLT

XML Stylesheet Language for Transformations.

XVU

DSS XML Validation Utility