305-CD-008-001

EOSDIS Core System Project

Release A SDPS Data Server Subsystem Design Specification for the ECS Project

July 1995

Hughes Information Technology Corporation
Landover, MD

Release A SDPS Data Server Subsystem Design Specification for the ECS Project

July 1995

Prepared Under Contract NAS5-60000
CDRL Item #046

SUBMITTED BY

Parag Ambardekar /s/ 7/28/95
Parag Ambardekar, Release A CCB Chairman Date
EOSDIS Core System Project

Hughes Information Technology Corporation
Landover, Maryland

305-CD-008-001

This page intentionally left blank.

Preface

This document is one of sixteen comprising the detailed design specifications of the SDPS and CSMS subsystem for Release A of the ECS project. A complete list of the design specification documents is given below. Of particular interest are documents number 305-CD-004, which provides an overview of the subsystems and 305-CD-018, the Data Dictionary, for those reviewing the object models in detail. A Release A SDPS and CSMS CDR Review Guide (510-TP-002) is also available.

The SDPS and CSMS subsystem design specification documents for Release A of the ECS Project include:

305-CD-004	Release A Overview of the SDPS and CSMS Segment System Design Specification
305-CD-005	Release A SDPS Client Subsystem Design Specification
305-CD-006	Release A SDPS Interoperability Subsystem Design Specification
305-CD-007	Release A SDPS Data Management Subsystem Design Specification
305-CD-008	Release A SDPS Data Server Subsystem Design Specification
305-CD-009	Release A SDPS Ingest Subsystem Design Specification
305-CD-010	Release A SDPS Planning Subsystem Design Specification
305-CD-011	Release A SDPS Data Processing Subsystem Design Specification
305-CD-012	Release A CSMS Segment Communications Subsystem Design Specification
305-CD-013	Release A CSMS Segment Systems Management Subsystem Design Specification
305-CD-014	Release A GSFC Distributed Active Archive Center Implementation
305-CD-015	Release A LaRC Distributed Active Archive Center Implementation
305-CD-016	Release A MSFC Distributed Active Archive Center Implementation
305-CD-017	Release A EROS Data Center Distributed Active Archive Center Implementation
305-CD-018	Release A Data Dictionary for Subsystem Design Specification
305-CD-019	Release A System Monitoring and Coordination Center Implementation

Object models presented in this document have been exported directly from CASE tools and in some cases contain too much detail to be easily readable within hard copy page constraints. The reader is encouraged to view these drawings on line using the Portable Document Format (PDF) electronic copy available via the ECS Data Handling System (ECS) at URL http://edhs1.gsfc.nasa.gov.

This document is a contract deliverable with an approval code 2. As such, it does not require formal Government approval, however, the Government reserves the right to request changes within 45 days of the initial submittal. Once approved, contractor changes to this document are handled in accordance with Class I and Class II change control requirements described in the EOS Configuration Management Plan, and changes to this document shall be made by Document Change Notice (DCN) or by complete revision.

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Drive
Landover, MD 20785

Abstract

The Data Server Subsystem has the responsibility for storing earth science and related data in a persistent fashion, providing search and retrieval access to this data, and supporting the administration of the data and the supporting hardware devices and software products. As part of its retrieval function, the subsystem also provides for the distribution of data electronically or on physical media.

Keywords: data, server, SDSRV, STMGT, DDSRV, DDIST, DBMS, robotics, distribution, subscription, hard, media, tapes, working, storage, archive, document, HTML, documents, types.

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Original
iii through xxiv	Original
1-1 through 1-2	Original
2-1 through 2-4	Original
3-1 through 3-16	Original
4-1 through 4-312	Original
5-1 through 5-74	Original
6-1 through 6-96	Original
7-1 through 7-62	Original
8-1 through 8-6	Original
9-1 through 9-4	Original
10-1 through 10-6	Original
11-1 through 11-4	Original
12-1 through 12-4	Original
A-1 through A-36	Original
AB-1 through AB-4	Original

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
305-CD-008-001	Final	July 1995	95-0471

This page intentionally left blank.

Contents

Preface

Abstract

Change Information Page

1. Introduction

1.1 Identification 1-1
1.2 Scope 1-1
1.3 Document Organization 1-1
1.4 Status and Schedule 1-2

2. Related Documents

2.1 Parent Documents 2-1
2.2 Applicable Documents 2-1
2.3 Information Documents Not Referenced 2-2

3. Subsystem Overview

3.1 Introduction and Context 3-1
3.2 Subsystem Overview 3-1
3.2.1 Subsystem Structure 3-1
3.2.2 Subsystem Design Rationale 3-10

4. SDSRV - Science Data Server CSCI

4.1 CSCI Overview 4-1
4.2 CSCI Context 4-1
4.3 CSCI Object Model 4-5
4.3.1 DsAdDataTypeCollector Class 4-26
4.3.2 DsAdDescriptor Class 4-27
4.3.3 DsAdLog Class 4-28
4.3.4 DsAdRequestInterface Class 4-28
4.3.5 DsCeCERES Class 4-30
4.3.6 DsClAction Class 4-30
4.3.7 DsClCollector Class 4-32

4.3.8 DsClCollectorVector Class 4-33
4.3.9 DsClCommand Class 4-34
4.3.10 DsClDescriptor Class 4-35
4.3.11 DsClESDTReference Class 4-37
4.3.12 DsClESDTReferenceCollector Class 4-42
4.3.13 DsClESDTReferenceVector Class 4-48
4.3.14 DsClNotificationReceiver Class 4-49
4.3.15 DsClQuery Class 4-51
4.3.16 DsClRequest Class 4-55
4.3.17 DsClRequestVector Class 4-57
4.3.18 DsClSubmittedRequest Class 4-58
4.3.19 DsClSubscription Class 4-62
4.3.20 DsClSubscriptionCollector Class 4-65
4.3.21 DsClTypeInfo Class 4-67
4.3.22 DsCnConfiguration Class 4-69
4.3.23 DsCnDSSConfiguration Class 4-71
4.3.24 DsCnDSSStartup Class 4-71
4.3.25 DsCoCombination Class 4-72
4.3.26 DsDbAccess Class 4-73
4.3.27 DsDbAttributeToTableVector Class 4-80
4.3.28 DsDbEngine Class 4-81
4.3.29 DsDbGranuleToDbVector Class 4-82
4.3.30 DsDbInterface Class 4-83
4.3.31 DsDeCoreValid Class 4-86
4.3.32 DsDeCoreValidVector Class 4-88
4.3.33 DsDeDD Class 4-89
4.3.34 DsDeDDVector Class 4-90
4.3.35 DsDeESDTDescriptor Class 4-91
4.3.36 DsDeESDTDescriptorSet Class 4-96
4.3.37 DsDeEvent Class 4-98
4.3.38 DsDeEventVector Class 4-100
4.3.39 DsDeMathOp Class 4-101
4.3.40 DsDeMetadataDef Class 4-102
4.3.41 DsDeMetadataDefVector Class 4-104
4.3.42 DsDeRange Class 4-105
4.3.43 DsDeScienceParameter Class 4-106
4.3.44 DsDeScienceParameterVector Class 4-108
4.3.45 DsDeSeries Class 4-109
4.3.46 DsDeService Class 4-110

4.3.47 DsDeServiceVector Class 4-111
4.3.48 DsDeStaticMetadata Class 4-113
4.3.49 DsDeStaticMetadataVector Class 4-114
4.3.50 DsDeValid Class 4-115
4.3.51 DsDeValidVector Class 4-116
4.3.52 DsDoReferencePaper Class 4-118
4.3.53 DsFactory Class 4-119
4.3.54 DsGeBrowseProduct Class 4-120
4.3.55 DsGeDynamicLibrary Class 4-121
4.3.56 DsGeECSDataProduct Class 4-122
4.3.57 DsGeESDT Class 4-124
4.3.58 DsGeESDTConfiguration Class 4-128
4.3.59 DsGeESDTDynamicLibrary Class 4-129
4.3.60 DsGeESDTEventTable Class 4-131
4.3.61 DsGeESDTServiceProvider Class 4-132
4.3.62 DsGeESDTWrapper Class 4-133
4.3.63 DsGeScienceData Class 4-134
4.3.64 DsGeSummaryProduct Class 4-137
4.3.65 DsGeTypeID Class 4-138
4.3.66 DsGvRadar Class 4-140
4.3.67 DsLiLIS Class 4-141
4.3.68 DsMdCatalog Class 4-141
4.3.69 DsMdMetadata Class 4-146
4.3.70 DsNmNMC Class 4-150
4.3.71 DsNpAncillary Class 4-150
4.3.72 DsNpCalibration Class 4-151
4.3.73 DsNpCorrelative Class 4-151
4.3.74 DsNpNonECSDataProduct Class 4-152
4.3.75 DsNpOA Class 4-153
4.3.76 DsNpPlatform Class 4-153
4.3.77 DsNpVersion0 Class 4-154
4.3.78 DsNsHistoricalData Class 4-154
4.3.79 DsNsMPR Class 4-155
4.3.80 DsNsNonECSDataProduct Class 4-156
4.3.81 DsNsProdPlans Class 4-156
4.3.82 DsNsProductionHistory Class 4-157
4.3.83 DsNsQAStatistics Class 4-158
4.3.84 DsNsScienceSoftwareArchivePackage Class 4-158
4.3.85 DsPrRadar Class 4-160

4.3.86 DsSbAction Class 4-161
4.3.87 DsSbActionBase Class 4-162
4.3.88 DsSbCallBackTimer Class 4-164
4.3.89 DsSbEvent Class 4-165
4.3.90 DsSbEventHandler Class 4-166
4.3.91 DsSbEventTimer Class 4-168
4.3.92 DsSbFactory Class 4-168
4.3.93 DsSbRegisteredEvent Class 4-169
4.3.94 DsSbSubscription Class 4-171
4.3.95 DsSbSubscriptionInterface Class 4-173
4.3.96 DsSbTimer Class 4-175
4.3.97 DsSd24BitImage Class 4-176
4.3.98 DsSd8BitImage Class 4-176
4.3.99 DsSdCSDT Class 4-177
4.3.100 DsSdGrid Class 4-179
4.3.101 DsSdImage Class 4-181
4.3.102 DsSdLookUpTable Class 4-183
4.3.103 DsSdPoint Class 4-183
4.3.104 DsSdRaw Class 4-185
4.3.105 DsSdSwath Class 4-185
4.3.106 DsSrClient Class 4-187
4.3.107 DsSrCommand Class 4-189
4.3.108 DsSrCommandBase Class 4-189
4.3.109 DsSrCommandInfo Class 4-191
4.3.110 DsSrConnection Class 4-192
4.3.111 DsSrQueuedConnection Class 4-195
4.3.112 DsSrRequest Class 4-196
4.3.113 DsSrRequestBase Class 4-198
4.3.114 DsSrRequestInfo Class 4-200
4.3.115 DsSrRequestVector Class 4-200
4.3.116 DsSrServer Class 4-201
4.3.117 DsSrSession Class 4-203
4.3.118 DsSrSubmittedRequestVector Class 4-205
4.3.119 DsSrWorkingCollection Class 4-206
4.3.120 DsSsSSMI Class 4-208
4.3.121 DsTmTMI Class 4-208
4.3.122 DsViVIRS Class 4-209
4.3.123 EosHdf24BitImage Class 4-210
4.3.124 EosHdf8BitImage Class 4-210

4.3.125 EosHdfGrid Class 4-211
4.3.126 EosHdfLUT Class 4-211
4.3.127 EosHdfPoint Class 4-212
4.3.128 EosHdfSwath Class 4-212
4.3.129 GlBinaryP Class 4-213
4.3.130 GlDateP Class 4-214
4.3.131 GlDoubleP Class 4-215
4.3.132 GlLongP Class 4-217
4.3.133 GlParameter Class 4-218
4.3.134 GlParameterList Class 4-220
4.3.135 GlStringP Class 4-222
4.3.136 GlTimeP Class 4-223
4.3.137 MSSLog Class 4-225
4.3.138 RWTPtrOrderedVector Class 4-225
4.4 CSCI Dynamic Model 4-226
4.4.1 SDSRV_Acquiring_an_ESDT 4-227
4.4.2 SDSRV_Asynchronous_Status_Updates 4-230
4.4.3 SDSRV_Auto-cancel_A_Subscription 4-232
4.4.4 SDSRV_Canceling_a_Subscription 4-234
4.4.5 SDSRV_Catalog_Deleting_a_Metadata_Entry 4-236
4.4.6 SDSRV_Catalog_Insert_Collection_Metadata 4-238
4.4.7 SDSRV_Catalog_Insertion_of_Metadata 4-240
4.4.8 SDSRV_Catalog_Search 4-242
4.4.9 SDSRV_Catalog_Updating_Metadata 4-244
4.4.10 SDSRV_Changing_A_Request_Priority 4-246
4.4.11 SDSRV_Client_Browsing 4-248
4.4.12 SDSRV_Client_Connecting_to_a_Data_Server 4-250
4.4.13 SDSRV_Client_Request_Submission 4-252
4.4.14 SDSRV_Client_Searching 4-254
4.4.15 SDSRV_Deleting_A_Queued_Request 4-256
4.4.16 SDSRV_Ending_Session_No_Active_Request 4-258
4.4.17 SDSRV_Fulfilling_a_One-time_Subscription 4-260
4.4.18 SDSRV_Fulfilling_Open_Ended_Subscription 4-262
4.4.19 SDSRV_Inserting_Composite_ESDT 4-264
4.4.20 SDSRV_Inserting_New_ESDT 4-266
4.4.21 SDSRV_Inserting_Single_ESDT 4-268
4.4.22 SDSRV_Instantiating_an_ESDT 4-270
4.4.23 SDSRV_Op_View_Queued_Requests 4-272
4.4.24 SDSRV_Registering_a_Subscribable_Event 4-274

4.4.25 SDSRV_Returning_List_of_Subscriptions 4-276
4.4.26 SDSRV_Server_Handling_A_Browse_Request 4-278
4.4.27 SDSRV_Server_Handling_A_Search_Request 4-280
4.4.28 SDSRV_Server_Request_Handling 4-282
4.4.29 SDSRV_Startup_of_a_Science_Data_Server 4-284
4.4.30 SDSRV_Submitting_a_Subscription 4-286
4.4.31 SDSRV_Unregistering_a_Subscribable_Event 4-288
4.4.32 SDSRV_Update_Server_Configuration 4-290
4.4.33 SDSRV_Updating_a_Subscription 4-292
4.4.34 SDSRV_Validating_Metadata 4-294
4.5 CSCI Structure 4-296
4.5.1 CSC Definitions 4-297
4.5.2 CSCI Dynamic Architecture 4-304
4.6 SDSRV CSCI Management and Operation 4-307
4.6.1 System Management Strategy 4-307
4.6.2 Operator Interfaces 4-309
4.6.3 Standard SDSRV Reports 4-311

5. DDSRV - Document Data Server CSCI 5-1

5.1 CSCI Overview 5-1
5.2 CSCI Context 5-1
5.3 CSCI Object Model 5-3
5.3.1 DsCdASCII Class 5-9
5.3.2 DsCdCSDT Class 5-10
5.3.3 DsCdHTML Class 5-12
5.3.4 DsCdKeyword Class 5-13
5.3.5 DsCdKeywordLocator Class 5-15
5.3.6 DsCdPDF Class 5-17
5.3.7 DsCdPostScript Class 5-18
5.3.8 DsCdRTF Class 5-20
5.3.9 DsCdTypeID Class 5-21
5.3.10 DsCsCSDT Class 5-21
5.3.11 DsCtAcquireCommand Class 5-22
5.3.12 DsCtClient Class 5-23
5.3.13 DsCtCommand Class 5-26
5.3.14 DsCtInsertCommand Class 5-27
5.3.15 DsCtRequest Class 5-29
5.3.16 DsCtSearchcommand Class 5-30

5.3.17 DsDoClient Class 5-33
5.3.18 DsDoCommand Class 5-34
5.3.19 DsDoRequest Class 5-34
5.3.20 DsDoServer Class 5-36
5.3.21 DsEsAlgorithmDescription Class 5-37
5.3.22 DsEsAlgorithmDescriptionTypeID Class 5-38
5.3.23 DsEsESDT Class 5-39
5.3.24 DsEsGuide Class 5-42
5.3.25 DsEsGuideTypeID Class 5-43
5.3.26 DsEsProductionPlan Class 5-44
5.3.27 DsEsProductionPlanTypeID Class 5-46
5.3.28 DsEsReferencePaper Class 5-47
5.3.29 DsEsReferencePaperTypeID Class 5-48
5.3.30 DsEsTypeID Class 5-49
5.3.31 DsGeCSDT Class 5-50
5.3.32 DsGeESDT Class 5-51
5.3.33 DsGeTypeID Class 5-51
5.3.34 DsSdCSDT Class 5-52
5.3.35 DsSdClient Class 5-52
5.3.36 DsSdCommand Class 5-53
5.3.37 DsSdESDT Class 5-54
5.3.38 DsSdESDT Class 5-54
5.3.39 DsSdRequest Class 5-55
5.3.40 DsSdServer Class 5-55
5.3.41 DsSdSession Class 5-56
5.3.42 DsSeIndexer Class 5-57
5.3.43 DsSeWWWServer Class 5-57
5.3.44 DsSvServer Class 5-58
5.4 CSCI Dynamic Model 5-60
5.4.1 Inserting a Document 5-61
5.4.2 Searching for a Document 5-65
5.4.3 Acquiring a Document 5-65
5.4.4 Fault Scenario: HTTP Connection Failure 5-65
5.5 CSCI Structure 5-67
5.5.1 DDSRV CSCs 5-67
5.5.2 DDSRV CI Processes 5-70
5.6 CSCI Management and Operation 5-71
5.6.1 System Management Strategy 5-71
5.6.2 Operator Interfaces 5-73

5.6.3 Standard DDSRV Reports 5-73

6. STMGT - Storage Management CSCI

6.1 CSCI Overview 6-1
6.2 CSCI Context 6-1
6.3 CSCI Object Model 6-5
6.3.1 DsCnConfiguration Class 6-8
6.3.2 DsSdAdvertisement Class 6-8
6.3.3 DsStArchive Class 6-9
6.3.4 DsStArchivedFile Class 6-12
6.3.5 DsStCDROM Class 6-14
6.3.6 DsStDirectory Class 6-17
6.3.7 DsStFax Class 6-20
6.3.8 DsStFile Class 6-20
6.3.9 DsStFileDirectory Class 6-22
6.3.10 DsStNetworkResource Class 6-24
6.3.11 DsStPhysicalResource Class 6-25
6.3.12 DsStPrinter Class 6-25
6.3.13 DsStPullConfig Class 6-26
6.3.14 DsStPullList Class 6-27
6.3.15 DsStPullMonitor Class 6-29
6.3.16 DsStRequestManager Class 6-33
6.3.17 DsStReservation Class 6-35
6.3.18 DsStResource Class 6-38
6.3.19 DsStResourceConfig Class 6-41
6.3.20 DsStResourceManager Class 6-42
6.3.21 DsStResourcePolicy Class 6-45
6.3.22 DsStResourceQueue Class 6-48
6.3.23 DsStResourceSchedule Class 6-50
6.3.24 DsStSCacheConfig Class 6-53
6.3.25 DsStSchedulingConfig Class 6-54
6.3.26 DsStStagingDataList Class 6-54
6.3.27 DsStStagingDisk Class 6-57
6.3.28 DsStStagingFile Class 6-59
6.3.29 DsStStagingMonitor Class 6-61
6.3.30 DsStStorageResource Class 6-64
6.3.31 DsStTape Class 6-65
6.3.32 GlLog Class 6-67
6.3.33 GlNotification Class 6-68

6.4 CSCI Dynamic Model 6-68
6.4.1 Aborting a Request for Service 6-69
6.4.2 Activating a Resource Reservation 6-69
6.4.3 Allocation of a Physical Resource, No Resource Available 6-69
6.4.4 Allocation of a Physical Resource, Resource Available 6-70
6.4.5 Inserting Data into the Archive 6-70
6.4.6 Retrieving Data from the Archive, Checksum Error 6-70
6.4.7 Retrieving Data from the Archive, File not in Cache 6-70
6.4.8 Canceling a Resource Reservation 6-71
6.4.9 Deallocation of a Physical Resource, Queued Requests 6-71
6.4.10 Deallocation of a Physical Resource, Imminent Reservation for Resource .. 6-71
6.4.11 Deletion of Pull Disk Data with Operations Confirmation 6-71
6.4.12 Deletion of Staging Disk Data 6-72
6.4.13 Setting the Operational State of a Tape Device 6-72
6.4.14 Submitting a Resource Reservation 6-72
6.5 CSCI Structure 6-87
6.5.1 CSC Definitions 6-87
6.5.2 CSCI Dynamic Architecture 6-90
6.6 CSCI Management and Operation 6-91
6.6.1 System Management Strategy 6-91
6.6.2 Operator Interfaces 6-93
6.6.3 Standard STMGT Reports 6-94

7. DDIST - Data Distribution CSCI

7.1 CSCI Overview 7-1
7.2 CSCI Context 7-1
7.3 CSCI Object Model 7-4
7.3.1 DsDdCDMedia Class 7-7
7.3.2 DsDdCDProcessor Class 7-7
7.3.3 DsDdDataItem Class 7-8
7.3.4 DsDdDistFile Class 7-9
7.3.5 DsDdDistList Class 7-10
7.3.6 DsDdDistRequest Class 7-11
7.3.7 DsDdDistRequestC Class 7-12
7.3.8 DsDdDistRequestS Class 7-13
7.3.9 DsDdElectronicMedia Class 7-16
7.3.10 DsDdLabeledMedia Class 7-18
7.3.11 DsDdMedia Class 7-21

7.3.12 DsDdOpsRequestC Class 7-23
7.3.13 DsDdPackingSlip Class 7-24
7.3.14 DsDdPrivRequest Class 7-25
7.3.15 DsDdPullMedia Class 7-27
7.3.16 DsDdPullProcessor Class 7-29
7.3.17 DsDdPushMedia Class 7-29
7.3.18 DsDdPushProcessor Class 7-32
7.3.19 DsDdRequestList Class 7-33
7.3.20 DsDdRequestManager Class 7-34
7.3.21 DsDdRequestManagerC Class 7-35
7.3.22 DsDdRequestManagerS Class 7-36
7.3.23 DsDdRequestProcessor Class 7-36
7.3.24 DsDdShippingLabel Class 7-38
7.3.25 DsDdTapeMedia Class 7-40
7.3.26 DsDdTapeProcessor Class 7-41
7.3.27 DsStResourceC Class 7-41
7.3.28 EcNotification Class 7-42
7.3.29 EcUtLogger Class 7-42
7.3.30 MsUsProfile Class 7-43
7.4 CSCI Dynamic Model 7-43
7.4.1 Distributed Creation of a New Distribution Request 7-44
7.4.2 Electronic Pull of Data 7-46
7.4.3 Electronic Push of Data 7-48
7.4.4 Physical Distribution to Tape 7-50
7.4.5 User Abort of a Request Waiting for a Device 7-52
7.4.6 Tape Fault 7-54
7.5 CSCI Structure 7-56
7.5.1 Distribution Products CSC 7-56
7.5.2 Distribution Client Interface CSC 7-57
7.5.3 Distribution Request Management CSC 7-57
7.6 CSCI Management and Operation 7-58
7.6.1 System Management Strategy 7-58
7.6.2 Operator Interfaces 7-60
7.6.3 Standard DDIST Reports 7-61

8. ACMHW - Access Control and Management HWCI

8.1 HW Design Drivers 8-1
8.1.1 Key Trade-off Studies and Prototypes 8-1
8.1.2 Sizing and Performance Analysis 8-2
8.1.3 Scalability, Evolvability, and Migration to Release B 8-2
8.2 HWCI Structure 8-2
8.2.1 HWCI Connectivity 8-2
8.2.2 HWCI Components 8-4
8.2.3 Failover and Recovery Strategy 8-5

9. WKSHW - Working Storage HWCI

9.1 HW Design Drivers 9-1
9.1.1 Key Trade-off Studies and Prototypes 9-1
9.1.2 Sizing and Performance Analysis 9-2
9.1.3 Scalability, Evolvability, and Migration to Release B 9-2
9.2 HWCI Structure 9-2
9.2.1 HWCI Connectivity 9-3
9.2.2 HWCI Component Description 9-3
9.2.3 Failover and Recovery Strategy 9-4

10. DRPHW - Data Repository HWCI

10.1 HWCI Design Rationale 10-1
10.1.1 Key Trades and Analysis 10-1
10.1.2 Scalability, Evolvability, and Migration to Release B 10-2
10.2 HWCI Structure 10-3
10.2.1 HWCI Connectivity 10-4
10.2.2 HWCI Component Description 10-5
10.2.3 Failover and Recovery Strategy 10-6

11. DIPHW - Distribution and Ingest Peripheral Management HWCI

11.1 HWCI Design Rationale 11-1
11.1.1 Key Trades and Analysis 11-1
11.1.2 Scalability Strategies 11-2
11.2 HWCI Structure 11-2
11.2.1 HWCI Connectivity 11-2
11.2.2 HWCI Component Description 11-3
11.2.3 Failover and Recovery Strategy 11-3

12. DDSRVHW - Document Data Server HWCI

12.1 HW Design Drivers 12-1
12.1.1 Key Trade-off Studies and Prototypes 12-1
12.1.2 Scalability, Evolvability, and Migration to Release B 12-1
12.2 HWCI Structure 12-1
12.2.1 HWCI Connectivity 12-1
12.2.2 HWCI Components 12-2
12.2.3 Failover and Recovery Strategy 12-2

Figures

3.1-1. Data Server Subsystem Context Diagram 3-2
3.2.2. Subsystem Design Rationale 3-10
3.2.2.1-1. Encapsulation of COTS in the Data Server Reference Architecture 3-11
4.3-1. DsAdAdmin Object Model Diagram 4-6
4.3-2. DsClCollector Object Model Diagram 4-7
4.3-3. DsClESDT Object Model Diagram 4-8
4.3-4. DsClQuery_Subscription Object Model Diagram 4-9
4.3-5. DsClRequest Object Model Diagram 4-10
4.3-6. DsCnConfiguration Object Model Diagram 4-11
4.3-7. DsCsCSDT Object Model Diagram 4-12
4.3-8. DsDbWrapper Object Model Diagram 4-13
4.3-9. DsDeESDTDescriptor Object Model Diagram 4-15
4.3-10. DsGeECSDataProductsRelA Object Model Diagram 4-17
4.3-11. DsGeGeneralESDT Object Model Diagram 4-18
4.3-12. DsMdMetadata Object Model Diagram 4-19
4.3-13. DsNpNonECSDataProduct Object Model Diagram 4-20
4.3-14. DsNpNonECSScienceData Object Model Diagram 4-21
4.3-15. DsNsNonScienceESDT Object Model Diagram 4-22
4.3-16. DsSbSubscription Object Model Diagram 4-23
4.3-17. DsSrScienceServer Object Model Diagram 4-24
4.3-18. GlParameter Object Model Diagram 4-25
4.4-1. SDSRV_Acquiring_an_ESDT Dynamic Model 4-229
4.4-2. SDSRV_Asynchronous_Status_Updates Dynamic Model 4-231
4.4-3. SDSRV_Auto-Cancel_A_Subscription Dynamic Model 4-233
4.4-4. SDSRV_Canceling_a_Subscription Dynamic Model 4-235
4.4-5. SDSRV_Catalog_Deleting_a_Metadata_Entry Dynamic Model 4-237
4.4-6. SDSRV_Catalog_Insert_Collection_Metadata Dynamic Model 4-239

4.4-7. SDSRV_Catalog_Insertion_of_Metadata Dynamic Model 4-241
4.4-8. SDSRV_Catalog_Search Dynamic Model 4-243
4.4-9. SDSRV_Catalog_Updating_Metadata Dynamic Model 4-245
4.4-10. SDSRV_Changing_A_Request_Priority Dynamic Model 4-247
4.4-11. SDSRV_Client_Browsing Dynamic Model 4-249
4.4-12. SDSRV_Client_Connecting_to_a_Data_Server Dynamic Model 4-251
4.4-13. SDSRV_Client_Request_Submission Dynamic Model 4-253
4.4-14. SDSRV_Client_Searching Dynamic Model 4-255
4.4-15. SDSRV_Deleting_A_Queued_Request Dynamic Model 4-257
4.4-16. SDSRV_Ending_Session_No_Active_Request Dynamic Model 4-259
4.4-17. SDSRV_Fulfilling_a_One-time_Subscription Dynamic Model 4-261
4.4-18. SDSRV_Fulfilling_Open_Ended_Subscription Dynamic Model 4-263
4.4-19. SDSRV_Inserting_Composite_ESDT Dynamic Model 4-265
4.4-20. SDSRV_Inserting_New_ESDT Dynamic Model 4-267
4.4-21. SDSRV_Inserting_Single_ESDT Dynamic Model 4-269
4.4-22. SDSRV_Instantiating_an_ESDT Dynamic Model 4-271
4.4-23. SDSRV_Op_View_Queued_Requests Dynamic Model 4-273
4.4-24. SDSRV_Registering_a_Subscribable_Event Dynamic Model 4-275
4.4-25. SDSRV_Returning_List_Of_Subscriptions Dynamic Model 4-277
4.4-26. SDSRV_Server_Handling_A_Browse_Request Dynamic Model 4-279
4.4-27. SDSRV_Server_Handling_a_Search_Request Dynamic Model 4-281
4.4-28. SDSRV_Server_Request_Handling Dynamic Model 4-283
4.4-29. SDSRV_Startup_Of_A_Science_Data_Server Dynamic Model 4-285
4.4-30. SDSRV_Submitting_a_Subscription Dynamic Model 4-287
4.4-31. SDSRV_Unregistering_a_Subscribable_Event Dynamic Model 4-289
4.4-32. SDSRV_Update_Server_Configuration Dynamic Model 4-291
4.4-33. SDSRV_Updating_a_Subscription Dynamic Model 4-293
4.4-34. SDSRV_Validating_Metadata Dynamic Model 4-295
5.2-1. DDSRVContext Diagram 5-2
5.3-1. DsDoServer Object Model Diagram 5-5
5.3-2. DsDoDocumentServer Object Model Diagram 5-6
5.3-3. DsDoESDT Object Model Diagram 5-7
5.3-4. DsDoCSDT Object Model Diagram 5-8
5.4-1. DDSRV_Insert Dynamic Model 5-62
5.4-2. DDSRV_Search Dynamic Model 5-63
5.4-3. DDSRV_Acquire Dynamic Model 5-64
5.4-4. Fault Scenario: HTTP Connection Failure 5-66
6.2-1. STMGT_events Event Flow Diagram 6-2
6.2-2. CSCI Interfaces 6-4

6.3-1. dsst Object Model Diagram 6-6
6.3-2. dsstconfig Object Model Diagram 6-7
6.4-1. ABORT_REQUEST Dynamic Model 6-73
6.4-2. ACTIVATE_RESERVATION Dynamic Model 6-74
6.4-3. ALLOCATEPHYSRES_NORESAVAIL Dynamic Model 6-75
6.4-4. ALLOCATEPHYSRES_RESAVAIL Dynamic Model 6-76
6.4-5. ARCHIVE_INSERT Dynamic Model 6-77
6.4-6. ARCHIVE_RETRIEVE_CHECKSUMERROR Dynamic Model 6-78
6.4-7. ARCHIVE_RETRIEVE_NOCACHEHIT Dynamic Model 6-79
6.4-8. CANCEL_RESERVATION Dynamic Model 6-80
6.4-9. DEALLOCATEPHYSRES_QUEUED_REQUESTS Dynamic Model 6-81
6.4-10. DEALLOCATEPHYSRES_RESERVATION Dynamic Model 6-82
6.4-11. DELETE_PULLDISKDATA_CONFIRMED Dynamic Model 6-83
6.4-12. DELETE_STAGINGDISKDATA Dynamic Model 6-84
6.4-13. SET_TAPESTATE Dynamic Model 6-85
6.4-14. SUBMIT_RESERVATION Dynamic Model 6-86
7.2-1. DDIST Event Flow Diagram 7-2
7.2-2. DDIST CI Interfaces 7-3
7.3-1. DsDdInterfaceClasses Object Model Diagram 7-5
7.3-2. DsDdManager Object Model Diagram 7-6
7.4.1-1. Creating a Distribution Request 7-45
7.4.2-1. Electronic Pull of Data 7-47
7.4.3-1. Electronic Push of Data 7-49
7.4.4-1. Distribution via Tape 7-51
7.4.5-1. Abort a Request 7-53
7.4.6-1. Handling a Tape Fault 7-55
8.2-1. Access Control and Management HWCI Block Diagram 8-3
8.2.1-1. Data Server Network Connectivity 8-3
9.2-1. Working Storage HWCI Block Diagram 9-3
10.2-1. Data Repository HWCI Block Diagram 10-4
11.2-1. Distribution and Ingest Peripheral Management HWCI Block Diagram 11-2
12.2.1-1. Access Control and Management HWCI Block Diagram 12-2

Tables

3.1-1. Subsystem Interfaces 3-3
3.2.2.2-1. Data Server Subsystem System Management GUI 3-15
3.2.2.2-2. Data Server Subsystem Request Management GUI 3-16
4.2-1. SDSRV CI Interfaces 4-3
4.5-1. SDSRV's Components 4-296
4.6-1. SDSRV Error Categories 4-308
4.6-2. Science Data Server Subsystem Administration Management GUI 4-310
4.6.3-1. Standard Science Data Server Reports 4-311
5.2-1. DDSRVContextevents Event Flow Summary Table 5-3
5.5.1. DDSRV Components 5-67
5.6-1. DDSRV Error Categories 5-72
5.6.2-1. Document Data Server Management GUI 5-73
5.6.3-1. Standard Document Data Server Reports 5-73
6.2-1. STMGT_events Event Flow Summary Table 6-3
6.2-2. CSCI Interfaces 6-4
6.5-1. STMGT's Components 6-87
6.6.1.2-1. STMGT Error Categories 6-93
6.6.2-1. Storage Management GUI 6-94
6.6.3-1. Standard Storage Management Reports 6-94
7.2-1. DDIST Event Flow Summary Table 7-3
7.2-2. DDIST CI Interfaces 7-3
7.5-1. DDIST's Components 7-56
7.6.1.2-1. DDIST Error Categories 7-59
7.6.2-1. Data Distribution Management GUI 7-60
7.6.3-1. Standard Distribution Reports 7-61
8.2.2-1. Access Control and Management HWCI Component Descriptions 8-5
9.2.2-1. Working Storage HWCI Component Descriptions 9-4
10.2.2-1. Data Repository HWCI Component Descriptions 10-6
11.2.2-1. Distribution and Ingest Peripheral HWCI Component Descriptions 11-3
12.2.2-1. Access Control and Management HWCI Component Descriptions 12-2

Appendix. Requirements Trace A-1

Abbreviations and Acronyms AB-1

1. Introduction

1.1 Identification

This Release A SDPS Data Server Subsystem Design Specification for the ECS Project, Contract Data Requirement List (CDRL) Item 046, with requirements specified in Data Item Description (DID) 305/DV2, is a required deliverable under the Earth Observing System Data and Information System (EOSDIS) Core System (ECS), Contract NAS5-60000. This publication is part of a series of documents comprising the Science and Communications Development Office design specification for the Communications and System Management Segment (CSMS) and the Science and Data Processing Subsystem (SDPS) for Release A.

1.2 Scope

The Release A Data Server Subsystem Design Specification defines the design of the subsystem. It defines the Data Server Subsystem computer software and hardware architectural design, as well as subsystem design based on Level 4 requirements.

This document reflects the June 21, 1995 Technical Baseline maintained by the contractor configuration control board in accordance with the ECS Technical Direction No. 11, dated December 6, 1994.

1.3 Document Organization

The document is organized to describe the Release A Data Server Subsystem design as follows:

Section 1 provides information regarding the identification, scope, status, and organization of this document.

Section 2 provides a listing of the related documents, which were used as source information for this document.

Section 3 provides an overview of the Subsystem, focusing on the high-level design concept. This provides general background information to put Data Server into context.

Section 4 describes the design of the Science Data Server (SDSRV) CSCI.

Section 5 describes the design of the Document Data Server (DDSRV) CSCI.

Section 6 describes the design of the Storage Management (STMGT) CSCI.

Section 7 describes the design of the Data Distribution (DDIST) CSCI.

Section 8 describes the design of the Access Control and Management (ACMHW) HWCI.

Section 9 describes the design of the Working Storage (WKSHW) HWCI.

Section 10 describes the design of the Data Repository (DRPHW) HWCI.

Section 11 describes the design of the Distribution and Ingest Peripheral Management (DIPHW) HWCI.

Section 12 describes the design of the Document Data Server (DDSRVHW) HWCI.

The appendix provides the Level 4 mapping matrix for use in verifying requirements.

Abbreviations and Acronyms contains an alphabetized list of the definitions for abbreviations and acronyms used in this document.

1.4 Status and Schedule

This submittal of DID 305/DV2 meets the milestone specified in the Contract Data Requirements List (CDRL) of NASA Contract NAS5-60000. The submittal was reviewed during the SDPS Preliminary Design Review (PDR) and reflects changes to the design which resulted from that review. The PDR also triggered a number of follow up actions in response to Review Item Discrepancies (RID) the results of which have been incorporated into the Critical Design Review (CDR) version of this document.

2. Related Documents

2.1 Parent Documents

The parent document is the document from which the scope and content of this Data Server Subsystem Design Specification is derived.

194-207-SE1-001	System Design Specification for the ECS Project

2.2 Applicable Documents

The following documents are referenced within this SDPS Design Specification, or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

209-CD-001-001	Interface Control Document Between EOSDIS Core System (ECS) and the NASA Science Internet
209-CD-002-001	Interface Control Document Between EOSDIS Core System (ECS) and ASTER Ground Data System
209-CD-003-001	Interface Control Document Between EOSDIS Core System (ECS) and EOS-AM Project for AM-1 Spacecraft Analysis Software
209-CD-004-001	Data Format Control Document for the Earth Observing System (EOS) AM-1 Project Data Base
209-CD-005-002	Interface Control Document Between EOSDIS Core System (ECS) and Science Computing Facilities (SCF)
209-CD-006-002	Interface Control Document Between EOSDIS Core System (ECS) and National Oceanic and Atmospheric Administration (NOAA) Affiliated Data Center (ADC)
209-CD-007-002	Interface Control Document Between EOSDIS Core System (ECS) and TRMM Science Data and Information System (TSDIS)
209-CD-008-002	Interface Control Document Between EOSDIS Core System (ECS) and the Goddard Space Flight Center (GSFC) Distributed Active Archive Center (DAAC)
209-CD-009-002	Interface Control Document Between EOSDIS Core System (ECS) and the Marshall Space Flight Center (MSFC) Distributed Active Archive Center (DAAC)
209-CD-011-002	Interface Control Document Between EOSDIS Core System (ECS) and the Version 0 System
305-CD-003-002	Communications and System Management Segment (CSMS) Design Specification for the ECS Project
308-CD-001-004	Software Development Plan for the ECS Project

313-CD-004-001	Release A CSMS/SDPS Internal Interface Control Document for the ECS Project
423-41-03	Goddard Space Flight Center, EOSDIS Core System (ECS) Contract Data Requirements Document

2.3 Information Documents Not Referenced

The following documents, although not referenced herein and/or not directly applicable, do amplify and clarify the information presented in this document. These documents are not binding on the content of the SDPS Design Specifications.

205-CD-002-002	Science User's Guide and Operations Procedure Handbook for the ECS Project. Part 4: Software Developer's Guide to Preparation, Delivery, Integration, and Test with ECS
206-CD-001-002	Version 0 Analysis Report for the ECS Project
209-CD-010-001	Interface Control Document Between EOSDIS Core System (ECS) and the Langley Research Center (LaRC) Distributed Active Archive Center (DAAC) Draft
194-302-DV2-001	ECS Facilities Plan for the ECS Project
101-303-DV1-001	Individual Facility Requirements for the ECS Project, Preliminary
194-317-DV1-001	Prototyping and Studies Plan for the ECS Project
318-CD-003-XXX	Prototyping and Studies Progress Report for the ECS Project (monthly)
333-CD-003-001	SDP Toolkit Users Guide for the ECS Project
601-CD-001-003	Maintenance and Operations Management Plan for the ECS Project
604-CD-001-004	Operations Concept for the ECS Project: Part 1 -- ECS Overview
101-620-OP2-001	List of Recommended Maintenance Equipment for the ECS Project
194-703-PP1-001	System Design Review (SDR) Presentation Package for the ECS Project
193-801-SD4-001	PGS Toolkit Requirements Specification for the ECS Project
194-813-SI4-002	Planning and Scheduling Prototype Results Report for the ECS Project
194-813-SI4-003	DADS Prototype One FSMS Product Operational Evaluation
194-813-SI4-004	DADS Prototype One STK Wolfcreek 9360 Automated Cartridge System Hardware Characterization Report
813-RD-009-001	DADS Prototype Two Multi-FSMS Product Integration Evaluation
828-RD-001-002	Government Furnished Property for the ECS Project
193-WP-118-001	Algorithm Integration and Test Issues for the ECS Project
193-WP-611-001	Science-based System Architecture Drivers for the ECS Project, Revision 1.0
193-WP-623-001	ECS Evolutionary Development White Paper
194-WP-901-002	EOSDIS Core System Science Information Architecture, White Paper, Working Paper

194-WP-902-002	ECS Science Requirements Summary, White Paper, Working Paper
194-WP-914-001	CORBA Object Request Broker Survey for the ECS Project, White Paper, Working Paper
194-WP-918-001	DADS Prototype One FSMS Product Operational Evaluation, White Paper, Draft Report
194-WP-925-001	Science Software Integration and Test, White Paper, Working Paper
420-WP-001-001	Maximizing the Use of COTS Software in the SDPS SDS Software Design, White Paper
193-TP-626-001	GCDIS/UserDIS Study ECS Technical Paper, Draft 0.2
194-TP-266-002	Data Distribution Architecture Logical Object Model (LOM) for the ECS Project, Version 2.01
194-TP-267-001	Data Server Architecture Logical Object Model (LOM) for the ECS Project, Version 2.00
194-TP-313-001	ECS User Characterization Methodology and Results
194-TP-316-002	Data Compression Study for the ECS Project
194-TP-548-001	User Scenario Functional Analysis [for the ECS Project]
194-TP-569-001	PDPS Prototyping at ECS Science and Technology Laboratory, Progress Report #4
222-TP-003-006	Release Plan Content Description for the ECS Project
430-TP-001-001	SDP Toolkit Implementation with Pathfinder SSM/I Precipitation Rate Algorithm, Technical Paper
440-TP-001-001	Science Data Server Architecture Study [for the ECS Project]
none	Hughes Training, Inc., ECS User Interface Style Guide, White Paper, Version 4.0
423-16-01	Goddard Space Flight Center, Data Production Software and Science Computing Facility (SCF) Standards and Guidelines
423-41-02	Goddard Space Flight Center, Functional and Performance Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) Core System
540-022	Goddard Space Flight Center, Earth Observing System (EOS) Communications (Ecom) System Design Specification
560-EDOS-0211.0001	Goddard Space Flight Center, Interface Requirements Document Between EDOS and the EOS Ground System (EGS)

This page intentionally left blank.

3. Subsystem Overview

3.1 Introduction and Context

This subsystem has the responsibility for storing earth science and related data in a persistent fashion, providing search and retrieval access to this data, and supporting the administration of the data and the supporting hardware devices and software products. As part of its retrieval function, the subsystem also provides for the distribution of data electronically or on physical media.

Figure 3.1-1 shows the subsystem context.

- The subsystem advertises its data types and the services it provides against this data with the SDPS Interoperability Services.
- It stores data received through ingest by the SDPS Ingest Subsystem, or resulting from processing in the SDPS Data Processing Subsystem, as well as historic data from FOS.
- It may issue production requests into the Planning Subsystem, and acquisition requests into a DAR processing subsystem, as a result of data requests issued by Data Server clients. It also supports the SDPS Planning Subsystem by storing data availability schedules.
- It accepts data search and access requests from any SDPS subsystem or other segment provided that they are directed specifically at the data objects managed by the given instance of the data server subsystem (all other search and access requests would be funneled through the DIM and/or LIM services of the SDPS Data Management Subsystem)
- It provides the data resulting from these access requests through electronic transfer or on physical media. The subsystem can also provide references to this data in the Universal Reference format instead (as described in this documents Overview volume, 305-CD-004-01).
- It interfaces with MSS/SMC to provide subsystem status and log information.

3.2 Subsystem Overview

3.2.1 Subsystem Structure

The Data Server subsystem is decomposed into four CSCIs and four HWCIs. All eight of these CIs are represented in Release A Data Server Subsystem configuration.

- Science Data Server CSCI (SDSRV) is a software component. It is responsible for managing the access to and storage of a collection of earth science and related data (i.e., ancillary data), except documents.
- The Document Data Server CSCI (DDSRV) is a software component. It manages document data. A separate document server is prompted by some of the unique aspects of document management, and the desire to provide an off-the-shelf solution for Release A. It is likely that DDSRV and SDSRV will be integrated into a single CSCI in a later release, by taking advantage of advances in DBMS technology.

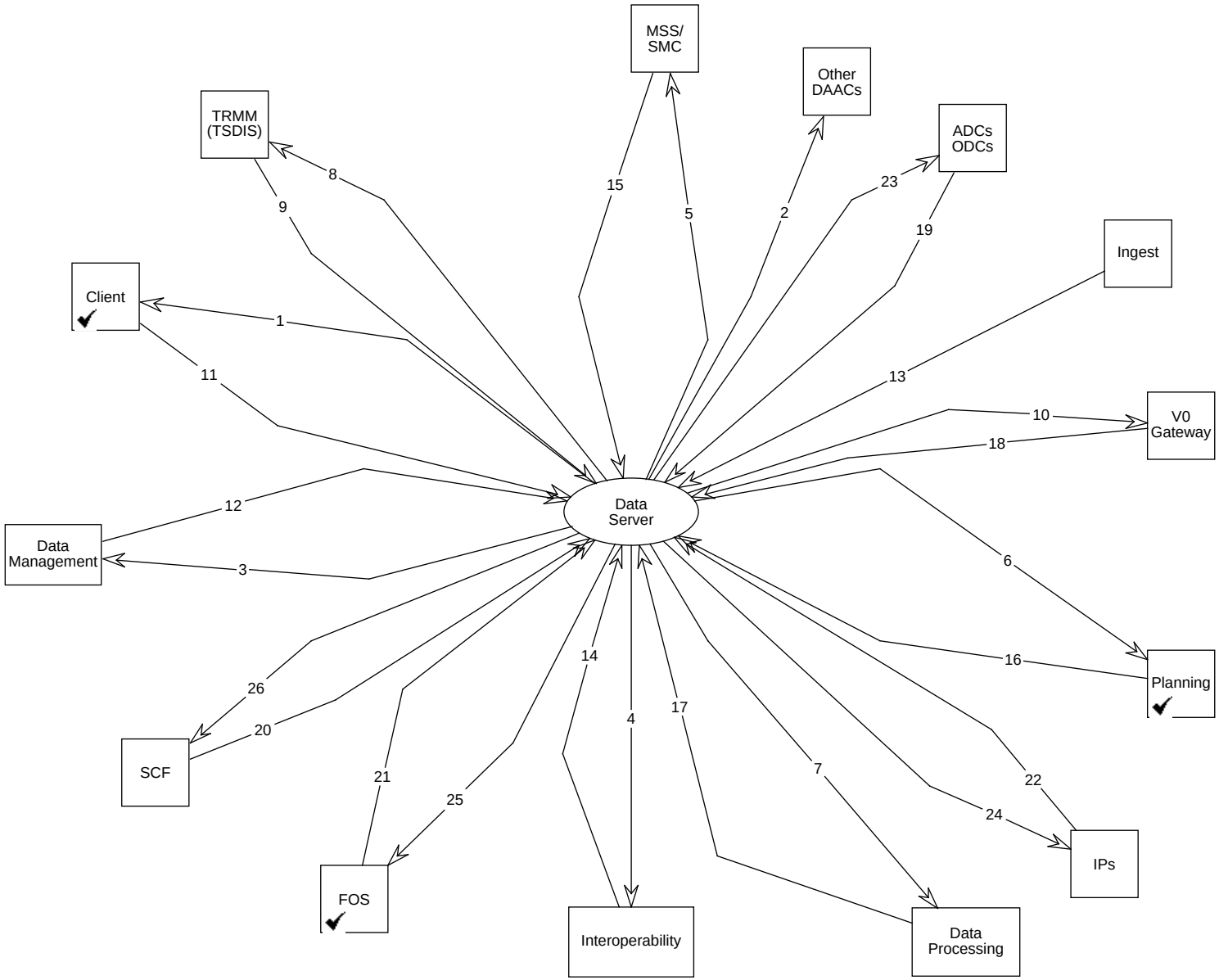


Figure 3.1-1. Data Server Subsystem Context Diagram

Table 3.1-1. Subsystem Interfaces (1 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
1	Data Server	Client *	Results Set	medium-high	in response to request
1	Data Server	Client *	Session Mgmt responses	low	in response to request
1	Data Server	Client *	Notifications	low	in response to subscription
2	Data Server	DAACs	Standard Products	high	as required
2	Data Server	DAACs	Metadata	medium-high	as required
2	Data Server	DAACs	Ancillary Data	high	as required
2	Data Server	DAACs	Correlative Data	high	as required
2	Data Server	DAACs	Calibration Data	high	as required
2	Data Server	DAACs	Documents	medium	as required
2	Data Server	DAACs	Orbit/Attitude Data	medium	as required
2	Data Server	DAACs	Data Availability Schedules	medium	as required
2	Data Server	DAACs	Algorithms	high	as required
2	Data Server	DAACs	Special Products	high	as required
2	Data Server	DAACs	L0 Data	high	as required
2	Data Server	DAACs	QA Data	medium	as required
3	Data Server	Data Mgmt	Session Mgmt Response	low	in response to request
3	Data Server	Data Mgmt	Search Results	medium-high	in response to request
3	Data Server	Data Mgmt	Schema	medium	as required
3	Data Server	Data Mgmt	data dictionary	medium	as required
4	Data Server	Interoperability	Advertisement	low	when service/data changes
4	Data Server	Interoperability	Subscription	low	as required
5	Data Server	MSS/SMC	Status	low	in response to request
5	Data Server	MSS/SMC	Logs	low	in response to request
6	Data Server	Planning	subscription notice	low	at fulfillment of subscription
6	Data Server	Planning	data query response (granule information)	medium	in response to request
7	Data Server	Processing	Standard Products	high	in response to staging request
7	Data Server	Processing	meta data	medium	in response to staging request

Table 3.1-1. Subsystem Interfaces (2 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
7	Data Server	Processing	L0 Data	high	in response to staging request
7	Data Server	Processing	Ancillary Data	high	in response to staging request
7	Data Server	Processing	Calibration Data	medium	in response to staging request
7	Data Server	Processing	Orbit/Attitude Data	medium	in response to staging request
7	Data Server	Processing	Algorithms	high	in response to staging request
8	Data Server	TRMM (TSDIS)	Metadata	medium - 2 day's worth of archived data	daily
8	Data Server	TRMM (TSDIS)	Ancillary Data	high - enough to support 1 day of processing and 2 day's worth of reprocessing (~10 Gbytes)	frequency depends on data set
8	Data Server	TRMM (TSDIS)	Calibration Data	medium	included in ancillary data
8	Data Server	TRMM (TSDIS)	Correlative Data	medium	included in ancillary data
8	Data Server	TRMM (TSDIS)	Documents	low	as required
8	Data Server	TRMM (TSDIS)	Data Products	2 days worth of archived data (~50 Gbyte)	daily
8	Data Server	TRMM (TSDIS)	Data Availability Notice	low	daily
8	Data Server	TRMM (TSDIS)	Status	low	as required
9	TRMM (TSDIS)	Data Server	Product Orders	low	as requested
9	TRMM (TSDIS)	Data Server	Subscriptions	low	as requested
10	Data Server	Version 0 Gateway	Results Set	medium-high	in response to request
10	Data Server	Version 0 Gateway	Session Mgmt responses	low	in response to request
10	Data Server	Version 0 Gateway	Product Request Status	low	as required
*23	Data Server	ADCs/ODCs	Metadata	medium	as required
*23	Data Server	ADCs/ODCs	Calibration Data, Correlative Data, Documents	medium-high	as required
*23	Data Server	ADCs/ODCs	Algorithms	medium-high	as required

Table 3.1-1. Subsystem Interfaces (3 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
*23	Data Server	ADCs/ODCs	Data Products	high	as required
*23	Data Server	ADCs/ODCs	Schedule Adjudication data	low	as required
*23	Data Server	ADCs/ODCs	Status	low	as required
*24	Data Server	IPs	Ancillary Data	medium	as required (Media Delivery to ASTER GDS
*24	Data Server	IPs	Correlative Data	medium	as required (Media Delivery to ASTER GDS
*24	Data Server	IPs	DARs	low	frequency dependent on user input (ASTER only)
*24	Data Server	IPs	Level 0 - Level 4 Products	high	dependent on user input (Media Delivery to ASTER GDS
*24	Data Server	IPs	Metadata	low	dependent on user input
*24	Data Server	IPs	Orbit/Attitude Data	low-medium	infrequent (only for s/c anomalies (for ASTER))
*24	Data Server	IPs	Schedule Adjudication Data	low	as required
*24	Data Server	IPs	Status	low	as required
*24	Data Server	IPs	Documents	low	as required
*24	Data Server	IPs	Calibration Data	medium	as required
*25	Data Server	FOS	Quick Look Images	medium	in response to request
*25	Data Server	FOS	Historic Data	medium	in response to request
*25	Data Server	FOS	DARs	low	as required
*26	Data Server	SCF	Status	low	as required
*26	Data Server	SCF	Metadata/updates	low	as required
*26	Data Server	SCF	Calibration data	medium	as required
*26	Data Server	SCF	Correlative data	medium	as required
*26	Data Server	SCF	Documents	low	as required
*26	Data Server	SCF	Algorithms/updates	medium	as required
26	Data Server	SCF	Standard Products	medium	daily as required for QA
11*	Client	Data Server	Search Requests	low	as requested
11*	Client	Data Server	Access Requests	low	as requested

Table 3.1-1. Subsystem Interfaces (4 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
11*	Client	Data Server	Session Mgmt Requests	low	as requested
11*	Client	Data Server	Subscriptions	low	as requested
12	Data Mgmt	Data Server	Search requests	low	in response to request
12	Data Mgmt	Data Server	Session Mgmt Requests	low	in response to request
12	Data Mgmt	Data Server	Access requests	low	in response to request
13	Ingest	Data Server	Standard Products	high	on reception
13	Ingest	Data Server	Metadata	medium	on reception
13	Ingest	Data Server	Ancillary Data	high	on reception
13	Ingest	Data Server	Correlative Data	high	on reception
13	Ingest	Data Server	Calibration Data	medium	on reception
13	Ingest	Data Server	Documents	medium	on reception
13	Ingest	Data Server	Orbit/Attitude Data	medium	on reception
13	Ingest	Data Server	Data Availability Schedules	medium	on reception
13	Ingest	Data Server	Algorithms	high	on reception
13	Ingest	Data Server	Special Products	high	on reception
13	Ingest	Data Server	L0 Data	high	on reception
13	Ingest	Data Server	Quick Look Data	medium	on reception
13	Ingest	Data Server	QA Data	medium	on reception
13	Ingest	Data Server	Resource Allocation / Deallocation Requests	low	on reception
14	Interoperability	Data Server	Notification	low	in response to subscription
15	MSS/SMC	Data Server	Status requests	low	in response to request
15	MSS/SMC	Data Server	Log Requests	low	in response to request
16	Planning	Data Server	candidate plans /active plans	medium	as required for archiving
16	Planning	Data Server	data query	low	as required
16	Planning	Data Server	Subscription Request	low	as required
17	Processing	Data Server	access requests	low	as required for processing
17	Processing	Data Server	standard products	high	as required for processing
17	Processing	Data Server	metadata	medium	as required for processing

Table 3.1-1. Subsystem Interfaces (5 of 5)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
17	Processing	Data Server	Science Algorithms	low	as requested
17	Processing	Data Server	QA Data	medium	as required for processing
18	Version 0	Data Server	Inventory	low	as required
18	Version 0	Data Server	Guide	low	as required
18	Version 0	Data Server	Browse data	medium	as required
18	Version 0	Data Server	Dependent Valids	low	as required
*19	ADCs/ODCs	Data Server	Product Requests	low	frequency dependent on user input
*19	ADCs/ODCs	Data Server	Status	low	as required
20	SCF	Data Server	QA Data request	low	as required
20	SCF	Data Server	QA Data Subscription	low	as required
*21	FOS	Data Server	historic data	low	as required for archiving
*21	FOS	Data Server	DAR status	low	when status changes
*21	FOS	Data Server	Status Info	low	in response to request
*21	FOS	Data Server	Acquisition Plan Schedule	medium	as required
*22	IPs	Data Server	DAR Status	low	frequency based on user input (ASTER only)

In the table, where an exact number is unavailable, the data volume is estimated as low (less than 1 MB), medium (between 1 MB and 1 GB), or high (greater than 1 GB) per use defined in the frequency column. The frequency information will be updated as the interfaces are fully defined.
Entries marked with (*) represent interfaces that are beyond Release A.

- The Storage Management CSCI (STMGT) is a software component. It manages and provides access to archive data. It also provides a stable interface to the other software within the data server subsystem to insulate them from future changes in storage technology of which ECS will want to take advantage.
- Data Distribution CSCI (DDIST) is a software component. It is responsible for providing the distribution services to the data server.
- Access Control and Management HWCI (ACMHW) is a hardware component. It is responsible for supporting the access to the data server.
- Working Storage HWCI (WKSHW) is a hardware component. It is responsible for supporting the needs for temporary and buffer storage.
- Data Repository HWCI (DRPHW) is a hardware component. It is responsible for storing and maintaining the permanent archive.

- Distribution and Ingest Peripherals HWCI (DIPHW) is a hardware component. It provides the hardware capabilities needed for data dissemination and ingest from the data server through various mechanisms.

The following paragraphs provide a brief summary of the purpose of each CSCI and HWCI.

Science Data Server CSCI

The Science Data Server CSCI (SDSRV CSCI) expresses the "data are data" concept of the architecture. By acting as the session manager and primary client interface to the data, the SDSRV provides access to the data server holdings using methods and interfaces which remain consistent regardless of how the data is stored. Access (searching, requesting) and instantiation (creation of objects) of the data server data holdings are represented as Earth Science Data Type classes (ESDTs). The Computer Science Data Type (CSDT) classes represent the physical organization and storage implementation of the ESDTs.

ESDT and CSDT are implemented by combining off the shelf DBMS technology with software developed to support the aspects that are unique to the ECS data and are not supported by the off the shelf software directly. For example, the DBMS (perhaps with third party software extensions which are also off-the-shelf) will support data types such as integer, floating point, string, arrays, images, time, and various kinds of spatial objects. However, when building database support for ECS products, software may need to be developed to support coordinate transformations which are not supported by the off-the-shelf software.

Information which resides in a DBMS is defined in a formal fashion. The definition is called a database schema. The DBMS uses it to interpret the queries and access requests it receives and execute them. If all support needed by the ECS science data were available off-the-shelf within the DBMS, the SDSRV software model would be very simple. It would consist, for the most part, of the DBMS. The data objects, i.e., the ESDT, are defined in the schema and are not visible in the software design of the SDSRV itself. The schema would implement the requirements of the ECS Core Meta Data Model (with product unique extensions). The Core Meta Data Model would not be reflected in the software design, only in the database schema.

In actual fact, the SDSRV needs to add software support for science data types, for example, when manipulating it for distribution, or to implement browse, subsetting, and subsampling. As a result, the SDSRV software design includes classes that represent portions of the ECS Core Meta Data Model as well as other software object classes which are needed to implement other aspects of the data server.

An example of this is the DsSrSession class. It provides clients with a uniform, stateful interface with the data server (rollback and suspension, operations on working collections), independent of the underlying servers for the ESDTs. The application of inheritance, polymorphism, and specialization constructs in the design enables the data server to evolve the structure and behavior of data holdings, as well as add new data types, supporting the design goal of being extensible.

Document Data Server CSCI

The allocation of document search and retrieval to the Document Data Server CSCI enhances the ability of the data server to respond to emerging technologies in this area. It is anticipated that the Document Data Server CSCI will consist of primarily COTS and public domain software components, and will provide World-Wide-Web style browsing and searching of document data types.

Storage Management CSCI

The data server must be able to respond to policy changes regarding archiving of data vs. creation of it when needed, or priorities in the allocation of resources, such as working storage, staging disks, and the variety of persistent storage in the Data Repository HWCI. This requires that the data server be "policy neutral". The facilities to adapt the physical storage of data in the data server to policy, while minimizing impact to availability, is provided by the Storage Management CSCI (STMGT CSCI). This CSCI provides an isolation layer between the search and access views of the archived data in the clients domain, and the physical storage mechanisms of the data internal to the archive. Through the use of unique data identifiers, the STMGT CSCI externalizes its data holdings to the SDSRV CSCI, while hiding the actual physical storage of its data. This allows the STMGT CSCI to optimize its archive storage and data migration strategies, while maintaining a consistent reference to the data for its clients.

The separation between the "push" and "pull" sides of the system into provider and consumer domains is essential for the data server to meet its requirements for highly available and reliable archival and distribution of data. The Data Distribution CSCI manages and provides access to the resources in the data "pull" side of operations. This facilitates the scaling and customization of distribution hardware, software, and operations for each operational site, while isolating the changes required to access and manage site specific requirements to this CSCI.

Data Distribution CSCI

The Data Distribution CSCI (DDIST) orchestrates the delivery of data to its end destination (e.g., user, DAAC). DDIST receives tasking, in the form of distribution requests, from the Science Data Server and Document Data Server CSCIs and coordinates the activities of the Storage Management CSCI in transferring the data to the media specified by the requester. DDIST also supports operator management of distribution by allowing operators to view, cancel, suspend/resume, and change the priorities of requests.

When a distribution request is received by DDIST, the requesting CSCI - either Science Data Server or Document Server - has already retrieved the data from the archive.

Distribution of this data can be via either electronic or physical media.

Electronic distribution may be requested via either push or pull. With push, DDIST uses network resources managed by Storage Management to transfer the data to a remote destination specified by the requester. For pull, the data is placed in an area managed by Storage Management, from which the request originator can retrieve the data.

Physical media distribution can be via 8mm tape, 6250 bpi 9-track tape, or CD-ROM. DDIST uses resources managed by Storage Management to transfer the data to the physical media.

Access Management HWCI

The Access Management Hardware Component (ACMHW HWCI) provides administration tools and functions and supports the majority of the Data Server's infrastructure requirements. This component manages logical data server access, maintains client sessions, and intelligently routes service requests to the appropriate Data Server subsystem component(s). This component is designed for flexibility and hardware expandability in order to manage the large number of sessions and processes in progress at any given time and to accommodate a heterogeneous group of client server interfaces.

Distribution and Ingest Peripherals HWCI

In order to support the heterogeneous methods of data distribution as well as electronic and hard media ingest of non time-critical data into the system, a Distribution and Ingest Peripherals HWCI (DIPHW CSCI) has been established. This HWCI does not replace the Ingest Subsystem but, instead focuses on ingestion management of electronic and physical media that does not require the same RMA as EDOS data. This component also buffers files for media generation and electronic deliver ("push") to users, supports conventional or special output processing (e.g. formatting, subsetting, compression, etc.) and provides file/disk management support for electronic user's data ("pull").

Working Storage HWCI

The Data Server must manage a large pool of very fast and fast random access storage as well as some fast sequential access storage devices. These devices are managed by the Working Storage HWCI (WKSHW). This component supplies the pool of storage used for temporary file and buffer storage within the Data Server and it provides hardware to manage the associated storage resources. Working Storage will house ingested data prior to migration to the Permanent Archive. In addition, it will contain "subscribed data" an pre/post processed data & products awaiting storage and/or distribution. This component will organize resources into storage tiers to provide the level of service/access required by the data user.

Data Repository HWCI

The Data Repository HWCI (DRPHW) provides the permanent data storage and maintenance functions for the Data Server. Multiple technologies can be instantiated into each DRPHW based on storage and access requirements. This component is designed for expandability and evolvability and will be sized to accommodate permanent storage and special access and storage requirements of specific data sets on site by site basis.

3.2.2 Subsystem Design Rationale

3.2.2.1 The Role of COTS

The motivation to provide a solution that is mostly COTS yet responsive to the requirements of an evolvable, heterogeneous system, provides the rationale for the data server reference architecture, shown in Figure 3.2.2.1-1. To support evolvability and technology insertion of the DBMS and FSMS data server components, the data server design encapsulates COTS solutions with wrapper classes. Isolation of vendor specific interfaces to a well-defined set of interface classes eases the

insertion of new technology by minimizing its impact on the design. Encapsulation of COTS through wrappers also accommodates the heterogeneity among site specific configurations in the data server, by loosely coupling the data server design with any one particular COTS solution.

In the following section, the role of the two major COTS components in the data server, the DBMS and the FSMS, will be discussed.

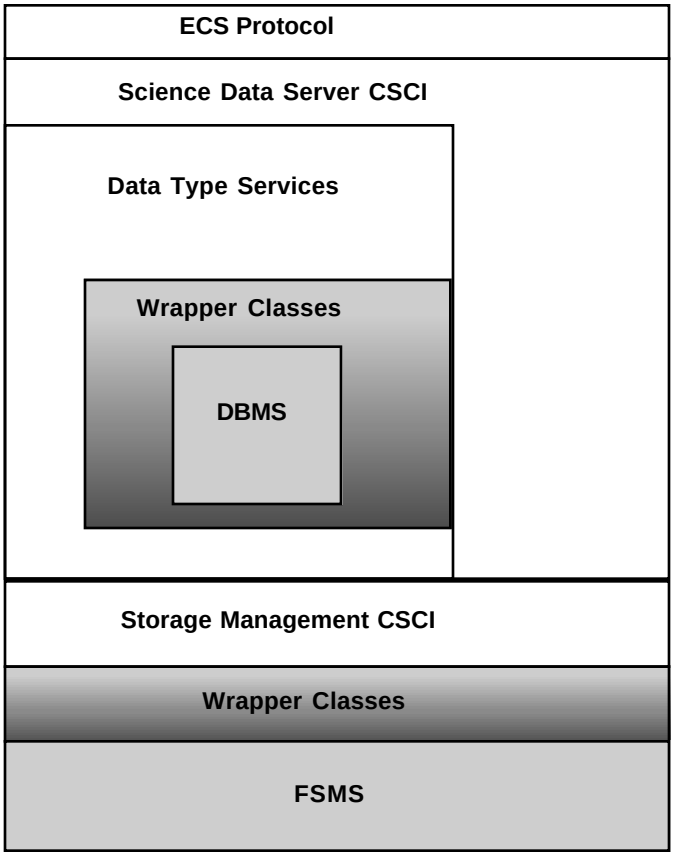


Figure 3.2.2.1-1. Encapsulation of COTS in the Data Server Reference Architecture

The Role of COTS Database Management Products

In addition to its traditional functions, the primary role of the DBMS in the data server will be to provide the storage, ad hoc retrieval services, and maintenance utilities of persistent data in support of the Earth Science Data Types, and to support information needed by the session and subscription services in order to maintain stateful, nested sessions with clients, and persistent storage of subscribed events and actions. In general, the traditional functions of a DBMS are to support:

- Data storage, retrieval, and update
- Schema services describing the structure and data items in the database
- Transaction control for ensuring performance of updates
- Concurrence control for multiple simultaneous update and retrieval

- Recovery services for restoring the database
- Authorization services for secure access to data and operations on the data
- Distributed client/server communication
- Integrity services for ensuring updates conform to the constraints of the application
- Utility services for the general maintenance of the database

These functions are common to commercially available DBMS products. Commercially available DBMS products generally fall under one of the following three categories, with some variations on, and extensions to, the preceding list of functions supported:

- Relational Database Management Systems (RDBMS). These systems are based on the relational data model. The data model is primarily developed on tabular data which has limited data types. This technology is the most mature and very popular, however, it does not allow the definition of complex data types and typically does not offer easy and efficient solutions to problems such as spatial data access. Some systems can be configured to interface with an external spatial index to provide efficient spatial searching.
- Object Oriented Database Management Systems (OODBMS). This technology allows the application to define complex objects with attributes and methods which operate on them. A data model can be defined in terms of class hierarchies using the object oriented paradigm. OODBMSs are primarily used in computer aided design and manufacturing (CAD/CAM) and computer aided software engineering (CASE) tools where the data types are very complex and contain many objects interconnected in a predetermined fashion. The primary access methodology is navigational through a programmatic interface.
- Object Relational Database Management Systems (ORDBMS). These are considered object relational because they support all the features of the relational data model and also partially support the object oriented paradigm. They allow the application to define complex objects with attributes and methods and also a class hierarchy. The classes in the data model map to tables.

Each of these product categories present the potential to implement data server functional requirements beyond those traditionally performed by a DBMS. The Sybase RDBMS has been selected to support Release A capabilities due to the established history of the Sybase product and the essential functionality this product provides the SDSRV (e.g. backup, SQL, availability of third party tools, etc.). The issue of spatial query performance is being addressed via the Spatial Query Server under development by Autometric Inc. Additional issues concerning query performance extensibility are being addressed by the vendor. Though Sybase may not be the final solution to the ECS metadata problem, it is apparent from trade studies and prototyping that there is no complete off-the-shelf solution at this point in time. Most solutions tested and specifically those that reached the evaluation “finals” had merits and problems associated with their implementation. Sybase does represent a viable solution that will effectively support Release A data rates and flows. The use of Sybase does not, in any way, break the objects paradigm being used by the SDSRV. The Release B team is continuing the database trade studies begun by Release A, and the encapsulation strategy being used by the development team will minimize the ramifications of any potential DBMS replacement in Release B and beyond.

The Role of COTS Storage Management Products

The ECS program will make use of COTS storage management products to perform the majority of the file and volume management functions required in the ECS archive. There are numerous File Storage Management Products (FSMS), (also known as Hierarchical Storage Management (HSM) products) on the market today. These products can generally be divided into three basic groups based on the method they use to interface with the Unix File System (UFS) and thus with the user. The groups are:

a. Unix Resident Products - which reside entirely within the confines of UFS. This allows native, RPC, API, and network-based (*nfs, ftp*, etc.) access to the data. These products use the UFS Index Node (Inode) structures to manage the data. Examples are EMASS's FileServ and EMC's EpochServ.

b. Separate File System Products - reside entirely outside of UFS. These products manage files in a separate proprietary system compatible with UFS in some ways. In almost all cases, access is exclusively via network-based access methods. An example of these products is the UniTree family.

c. Unix / Separate File System Hybrids - are products that use a separate file system for file management but are linked to UFS via the Virtual File System (VFS) Virtual Node (Vnode) in the Unix kernel. (This is the same point where *nfs* interfaces with UFS.) These products allow native, RPC, API and network-based access to file data but do not manage files using Inodes. Examples are Legent's OSM, EMASS's AMASS, Qstar's Mastermind, ANT's Metior, and LSC's Sam-FS.

FSMS products have primarily been designed to fill a long standing need in the user community. This need is to maximize the effective use of storage resources while minimizing cost. This is accomplished via the HSM paradigm. This paradigm assumes that as data ages in the system it becomes less frequently accessed. This lead to the Hot - to - Cold life cycle and a hierarchy of storage. When data enters a system it is “hot” and frequently accessed. Therefore it is stored on high speed secondary disk storage for rapid retrieval. As the data “cools” it is “migrated” to slower disk storage (e.g. magneto optical), then to tertiary storage devices such as magnetic tapes, and finally it may reside on a shelf in an “off-line” archive. FSMS products migrate files from secondary to tertiary to off-line storage and manage the location of these files either with Inodes or with a separate database. The location of the file is tied to the account of the user that owns the file. By using the user's path name in conjunction with the file name, this establishes a unique name for each file. The file still appears to be resident on secondary storage from a user's perspective. There is just a somewhat longer delay when the file is “staged” from tertiary storage back to secondary storage. Thus tertiary storage provides users with an almost limitless virtual disk for there files.

Unfortunately, the HSM Paradigm will not cost effectively fit into the ECS environment. HSM performance is based upon the frequency of data being located in an associated disk cache. Thus, these systems are designed to maximize “cache hits”. This requires secondary storage resources to equal 10 percent or more of all storage managed. In a 10 Terabyte or smaller system this may be feasible but in a multi-Petabyte system, like ECS, the cost of magnetic disk resources quickly becomes prohibitive. Another limitation, from an ECS perspective, is the path to file name association used by HSM systems to manage unique file names. This system requires a file to return to its root (e.g. Inode or directory position) before it can be modified or moved.

ECS requires an Archive Management Server (AMS). This system encapsulates the capabilities of an existing HSM or FSMS product via APIs and RPCs. The COTS software will still manage files and the associated media volumes and it will insure long term archive viability. The inherent data movement capabilities of the COTS software, however, will be overridden by an ECS cache management scheme. The ECS system will augment the existing capabilities of a COTS software package to fit an AMS environment.

The AMS Paradigm is not a virtual disk system. AMS relies on two general storage tiers: Working Storage and Permanent Archive. The working storage tier hold data that can be thought of as or hot. This tier will temporarily hold data either before or after processing, and as a buffer for ingestion and distribution. As system requirements grow other tiers of storage devices will augment the disk resources in working storage. The Permanent Archive will form the tertiary layer of the system and will store all data designated by the system. The encapsulated FSMS or HSM product will move the data between storage tiers and will insure the data remains error free. Encapsulation will also allow replacement of obsolete HSM products as newer products emerge. Data will not move through the system based on a user name and associated path. The goal is to serve data directly to a designated user's disk without moving it through a specific directory hierarchy. Thus this approach will support horizontal scaling by allowing multiple servers in a network environment to manage the Permanent Archive, and evolvability of both the archive and the technology that supports it.

The FSMS product selected for Release A is the Archival Management and Storage System (AMASS) product marketed by EMASS Inc. The VFS-linked separate file system design allows all Unix File System (UFS) access methods to be employed (e.g. ftp, rcp, uucp, nfs, RPC, native, etc.) while removing some of the limitations of the UFS. Primary among these is reliance on Unix Index Node (Inode) structures. AMASS maintains all inode information in database files rather than in associated disk structures. This minimizes or eliminates many of the file search problems inherent in large numbers of files in multiple directories. In addition, AMASS organizes files as groups of blocks which can be individually retrieved. This differs from UFS resident systems which require staging the entire file. AMASS utilizes a disk based I/O buffer for communications rate matching between disk and tape resources. The I/O buffer scheme fits the AMS paradigm of encapsulating HSM storage and data viability functions while allowing the Data Server Storage Management CSCI manage the associated Working Storage.

3.2.2.2 Layering of GUIs

The Data Server Subsystem must be operable and controllable. This is made difficult by the size and complexity of the Subsystem. Our approach to operator control is to “layer” our operator interfaces (GUIs). The subsystem level GUIs will provide normal operational control and insight into system operations. These views into the system are grouped into system management and request management functions. CI level information, both software and hardware, are available to the operator if he or she wants to “drill down” into the internals of the system, a request, or an active system user. The subsystem level GUIs are described here in Tables 3.2.2.2-1 and 3.2.2.2-2. Each of the CI GUI interfaces are described in the appropriate CI section.

Table 3.2.2.2-1. Data Server Subsystem System Management GUI

GUI	Description	Data	Operations
Data Server Subsystem Management	Primary Screen for Data Server System Management. Root screen for the following GUIs.	References to: System State, Log & Reports, Configuration, Client Management and Resource Management GUIs.	Logon Realizes and/or makes the GUIs visible.
System State GUI	Allows operators to manage the state of the overall Data Server.	Subsystem name Subsystem state (active, degraded, down)	Startup Shutdown Restart
Log & Reports GUI	Allows operators to view and manage DSS logs. This GUI will utilize the logging and reporting GUIs provided by MSS.	Logged information including status, errors, faults	View log contents Sort Generate Report (print)
Configuration GUI	Allows operators to manage the configuration of DSS component software.	Persistent Configuration information (file or database)	View Update
Client Management GUI	Allows operators to manage the DSS clients. This GUI will utilize the GUIs provided by MSS.	Client information including: name, email address, privilege information. (for further definition see MSS User Profile)	View Register Logon Unregister Update
Resource Management GUI	Allows operators to manage overall DSS resources.	Scope/Filter (DSS-wide, Process, HWCI, Client) Resource identified Location Description State Utilization data	Update State View

Table 3.2.2.2-2. Data Server Subsystem Request Management GUI

GUI	Description	Data	Operations
Data Server Request Management	Primary Screen for Data Server Requests. Also root screen for the following GUIs (drill-down support).	Request information: ID, CSCI, Process, State, Current Status, Client, Priority References to: Request, Log & Reports, Configuration, Client Management and Resource Management GUIs.	Logon Filters on: All, State, Client, CSCI, Process, Priority) View Requests Delete Requests Update Requests Realizes and/or makes the GUIs visible.
Request Lineage	Allows operators to determine the relationships between requests.	Request information: ID, CSCI, Process, State Current Status, Priority Child Request Parent Request	View Print See lineage Show Parent Show Child
Client Information	Allows operators to manage the DSS clients. This information is provided to support a “drill-down” information about a request. This GUI will utilize the GUIs provided by MSS.	Client information including: name, email address, privilege information. (for further definition see MSS User Profile)	View Register Logon Unregister Update
Process Resource Management	Allows operators to manage overall DSS resources utilized by a request. This information is provided to support a “drill-down” information about a request.	Scope/Filter (DSS-wide, Process, HWCI, Client [default]) Resource identified Location Description State Utilization data	Update State View

4. SDSRV - Science Data Server CSCI

4.1 CSCI Overview

The Science Data Server CSCI provides the interface via which clients request the functions offered by the data server subsystem. There may be one or several implementations of the Science Data Server CSCI (called Science Data Servers) at any given site. Each Science Data Server provides access to a collection of earth science and related data. The scope of data and services offered by a Science Data Server are defined in the following manner:

- The Science Data Server makes its data types and operations on those data types known to the ECS community by advertising itself to the Advertising Server (ADSRV, part of the Interoperability Subsystem, see Volume 6 of this design specification). The contents of Advertisements include the identifier of the specific science data server (the service provider), the name of the data type (ESDT) offering that service, and the name of the service being offered, as well as descriptions of the general services being offered for the collection and the required interface definition of the service.
- The Science Data Server uses formal descriptions of its ESDTs, including their attributes and operations, as a Data Server Descriptor. The Descriptor for an ESDT contains many pieces of information about the specific ESDT that are required for the ESDT to participate in the ECS. Data search and access operations are expressed in terms provided by this Descriptor. For example, searching for LIS03 data based on 'spatial coverage' and 'temporal coverage' is only possible if 'spatial coverage' and 'temporal coverage' have been defined in the LIS03 Descriptor; the definition would specify, for example, what types of input parameters can be used in the search. Requirements for this schema-type information contained within the Descriptor are laid down in the ECS Core Meta Data Model.
- The Science Data Server provides a description of the meaning of each ESDT, each attribute, and each operation or service. This information is an integral part of the database managed by the Science Data Server, is available for access by queries sent to a Data Server, and is made available for access by and input into the Data Dictionary Services CSCI (DDICT, part of the Data Management Subsystem, not a Release A component). The source of this information is also in the ESDT's Descriptor.

4.2 CSCI Context

Figure 4.2-1 provides the context diagram for the SDSRV CI. This context diagram shows the Release A interfaces to other ECS CSCIs.

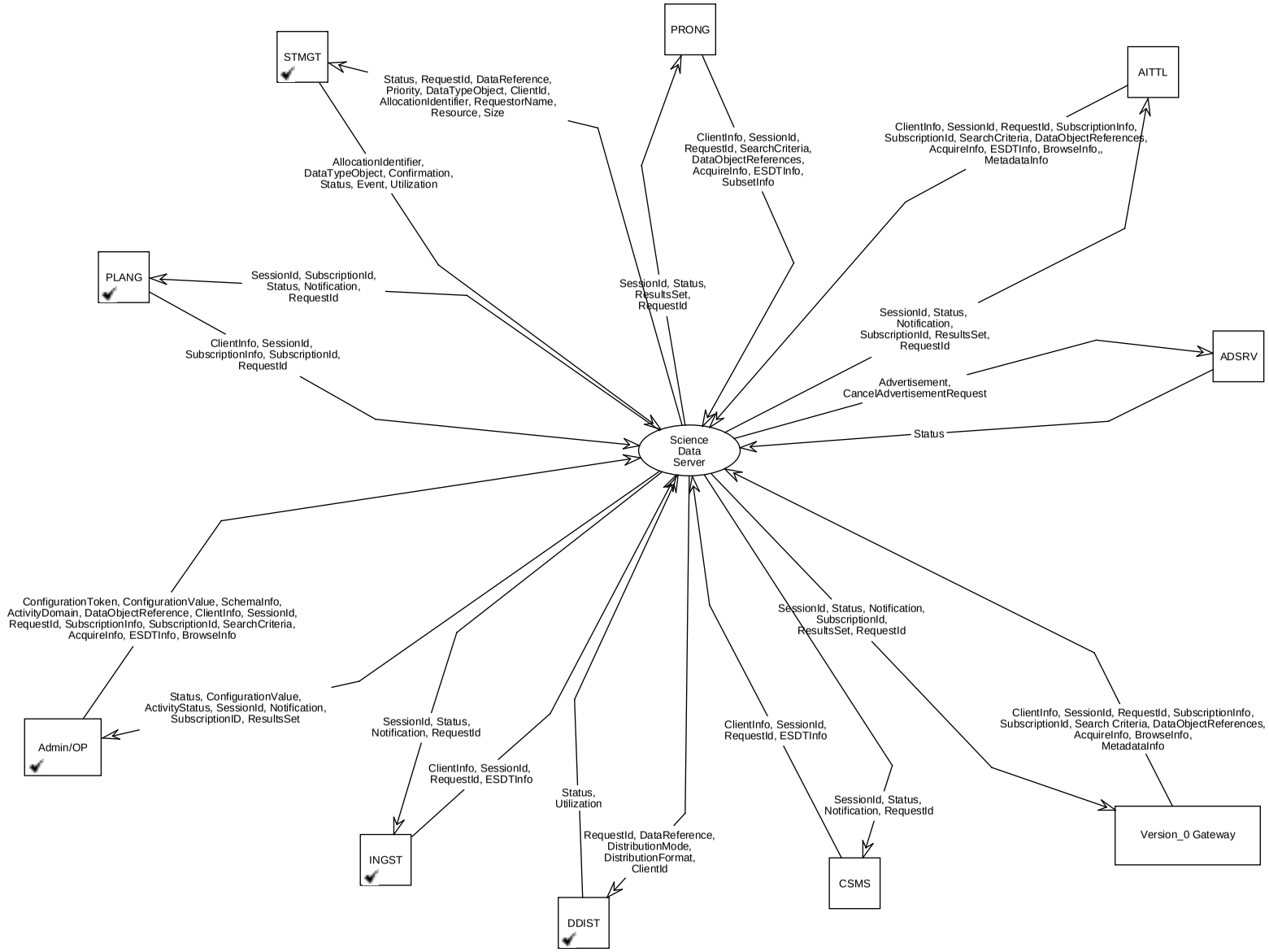


Figure 4.2-1. SDSRV Context Diagram

Table 4.2-1 provides a mapping for the SDSRV CI interfaces provided to other ECS CSCI's. The table defines the interfaces in terms of the SDSRV-provided classes (from the Client CSC), and the specific service to be used to accomplish that interface. The response (return parameter) from that member function is also included.

Table 4.2-1. SDSRV CI Interfaces (1 of 3)

Interface	Interface Input	Interface Output	Description
ConnectServer (Session)	DsClESDTReferenceCollector::DsClESDTReferenceCollector ()	*DsClESDTReferenceCollector	This interface is used by SDSRV clients to initiate a session with the SDSRV. The SDSRV will create a session for this client and return a reference to that session.
GetStatus (Session | Request)	DsClRequest::GetStatus()	GlStatus	This interface is used by SDSRV clients to obtain the current status of a specified session, or a specified request within that session. The SDSRV will return the status of the request.
Terminate (Session | Request)	DsClESDTReferenceCollector::~DsClESDTReferenceCollector ()	void	This interface is used by SDSRV clients to terminate the entire session with the SDSRV.
ChangePriority (Session | Request)	DsClRequest::SetPriority(DsTRequestPriority)	GlStatus	This interface is used by the SDSRV clients to alter the priority of a given request within a session. The SDSRV will respond with a status.
Notify (Session | Request)	none	GlNotification	This interface is used by the SDSRV to send notifications to its clients. The notification may be either a direct message (for example, information to be displayed within a dialogue box on the desktop) or an e-mail message (utilizing the CSS e-mail services).
Search	DsClESDTReferenceCollector::Search(DsClQuery)	GlStatus	This interface is used by SDSRV clients to search a collection of data objects. The search will either define or refine the domain of data objects that the client has in their context of the SDSRV holdings. The SDSRV will return a status of the search and populate the DsClESDTReferenceCollector with DsClESDTReference objects for each search "hit".
AddtoCollection	DsClESDTReferenceCollector::Insert(DsClESDTReference)	GlStatus	This interface is used by SDSRV clients to add data objects to the client’s current collection of data objects. The SDSRV will return the status of the collection modification.

Table 4.2-1. SDSRV CI Interfaces (2 of 3)

Interface	Interface Input	Interface Output	Description
GetResultsSet	DsClRequest::GetResults()	GlParameter List	This interface is used by SDSRV clients to assess the results of a completed request. A ResultsSet is a implemented as a GlParameterList.
Acquire	DsClESDTReferenceCollector::Submit(DsClRequest)	GlStatus	This interface is used by SDSRV clients to submit request for the acquire service of ESDT data objects. The client will submit information about the acquisition (e.g. other support files [QA Statistics, ProductionHistory, etc.]) as a GlParameterList within the request. The SDSRV will return a status of the acquisition request.
Insert	DsClESDTReferenceCollector::Submit(DsClRequest)	GlStatus	This interface is used by SDSRV clients to submit a request for the insert service of an ESDT. This is an interface which is expected to be rather limited in terms of availability to general users. The SDSRV will return the status of the insert request.
Browse	DsClESDTReference::Submit(DsClRequest)	GlStatus	This interface is used by SDSRV clients to submit a request for the browse service of an ESDT data object. The SDSRV will return the status of the browse request.
Change Configuration	DsAdRequestCollector::Update(GlParameter)	GlStatus	This interface is used by SDSRV administrator/operators to modify the configuration of the SDSRV. The administrator/operator will submit the configuration item as a token and the new value that the configuration item is to have. The SDSRV will return a status of the ChangeConfiguration service.
GetConfiguration	DsAdRequestCollector(GlParameter&)	GlParameter	This interface is used by SDSRV administrator/operators to retrieve the value of an SDSRV configuration item. The administrator/operator will submit the desired configuration item token and the SDSRV will return the value of that configuration item.

Table 4.2-1. SDSRV CI Interfaces (3 of 3)

Interface	Interface Input	Interface Output	Description
UpdateSchema	DsAdDescriptor::Update(GlParameterList)	GlStatus	This interface is used by SDSRV administrator/operators modify the existing data type schema. The administrator/operators will submit the new schema information, or a reference to where it is held. The SDSRV will update the schema and return the status of the UpdateSchema service.
MonitorActivity	DsAdRequestCollector::ListAllRequests()	DsClSRVCollector	This interface is used by SDSRV administrator/operators to monitor the activity of the SDSRV. The SDSRV will respond with a vector of all currently submitted requests.

4.3 CSCI Object Model

This section provides and describes the object model for the Science Data Server CSCI's design. The model is depicted in Figures 4.3-1 through 4.3-18. Each of the objects in this figure is described in the subsequent paragraphs. The Science Data Server Object model consists of a series of OMT class diagrams. The SDSRV CI's design has been decomposed into a set of class categories. A class category is a collection cooperating classes that fulfill a common set of responsibilities to other classes in the system. Each class category can be viewed from two perspectives: the public view and the implementation view. The public view is the interfaces (method signatures) of those classes that the users (other software classes, outside this class category) have access to, or will interface with. The implementation view is the entire set of classes that comprise the class category. This includes both those classes in the public view, as well as the collaborating classes used by those public classes. Each class category has at least one OMT class diagram. The public view is defined as those classes adorned as [PUBLIC]. Due to complexity, some class categories are modeled in more than one OMT class diagram.

General Notes: In the following object class descriptions, classes always inherit attributes, operations, and associations from their parent classes. Where all attributes in the current design are inherited from the parent class, this is indicated by the text "*All Attributes inherited from the parent class*" (and analogously for operations). If a derived class has additional attributes of its own, those new attributes are listed, but the attributes from the parent class are not repeated (analogously for operations).

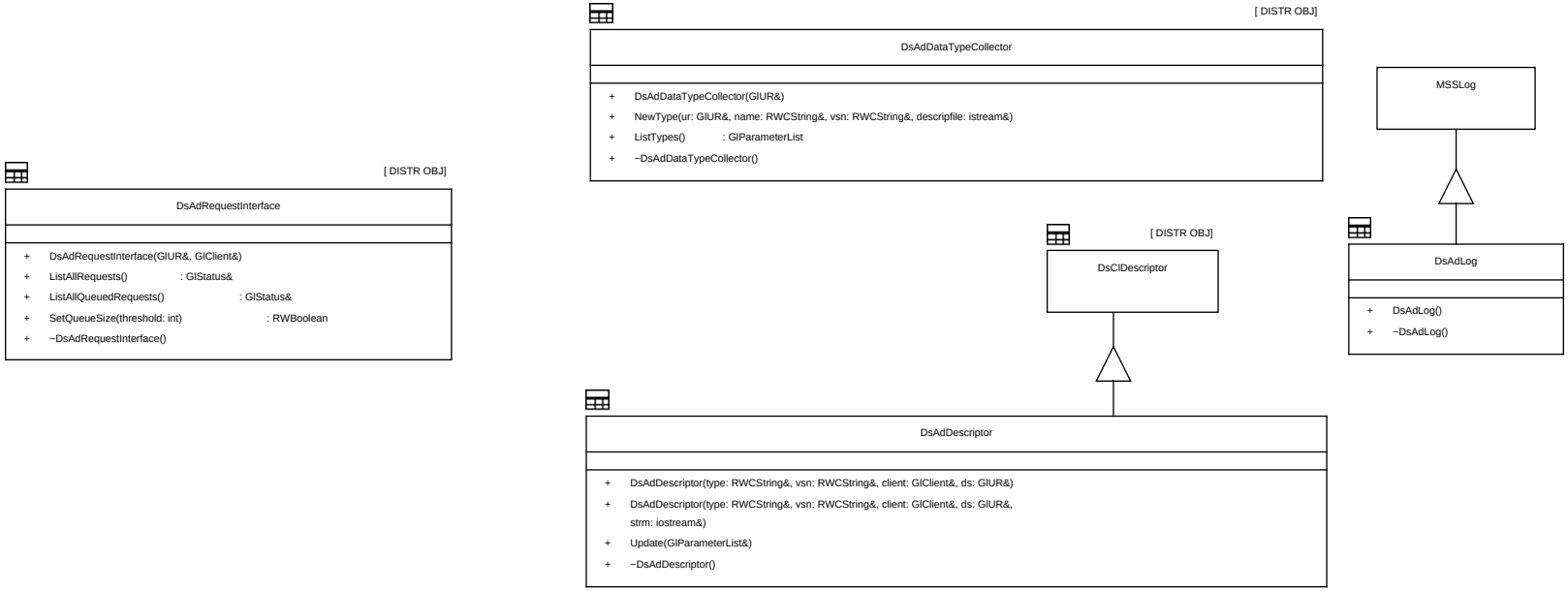


Figure 4.3-1. DsAdAdmin Object Model Diagram

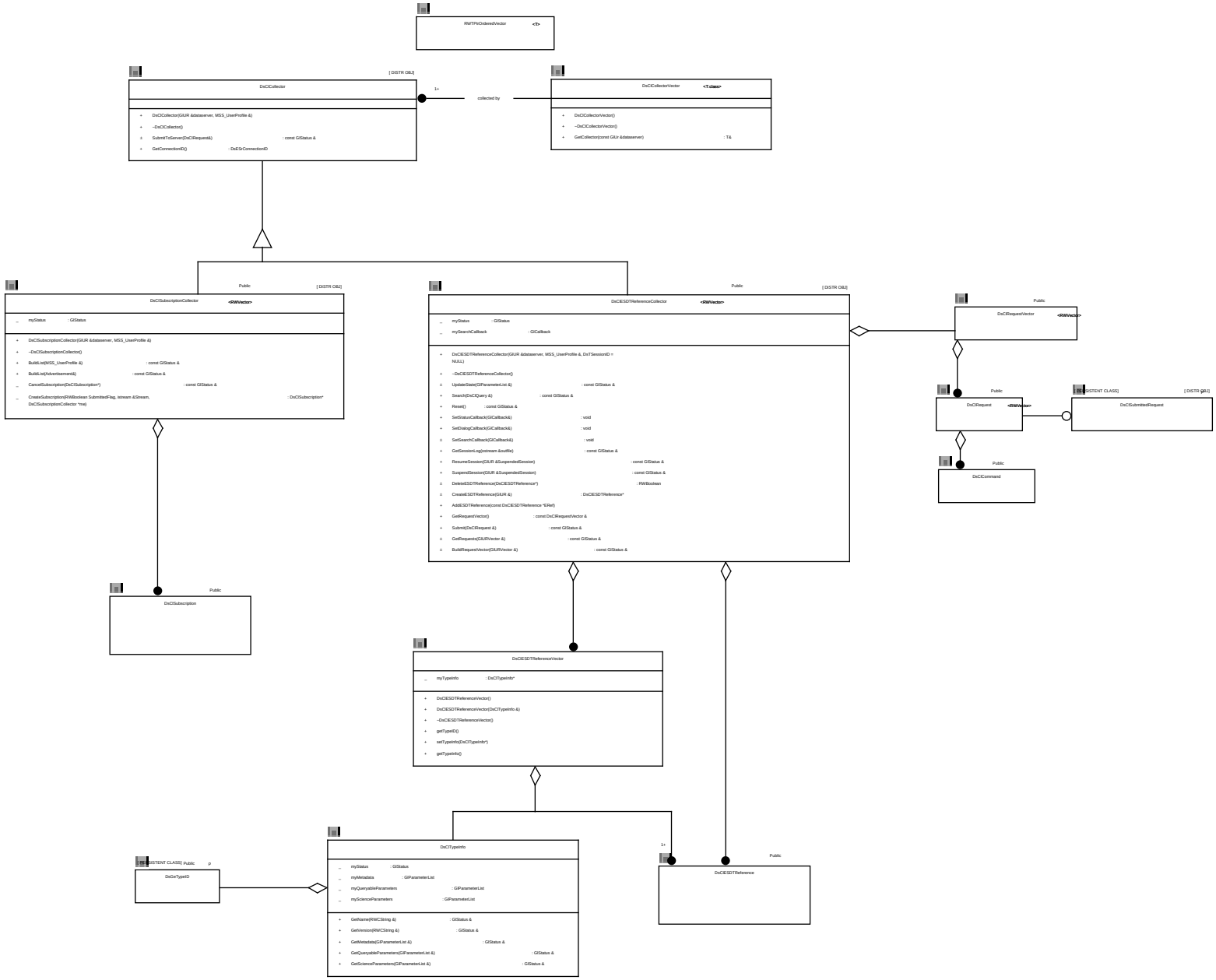
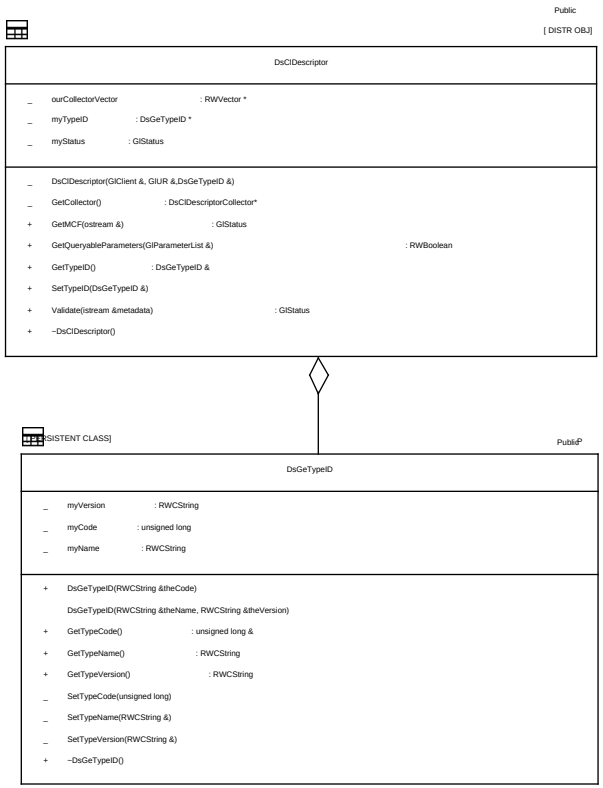


Figure 4.3-2. DsClCollector Object Model Diagram



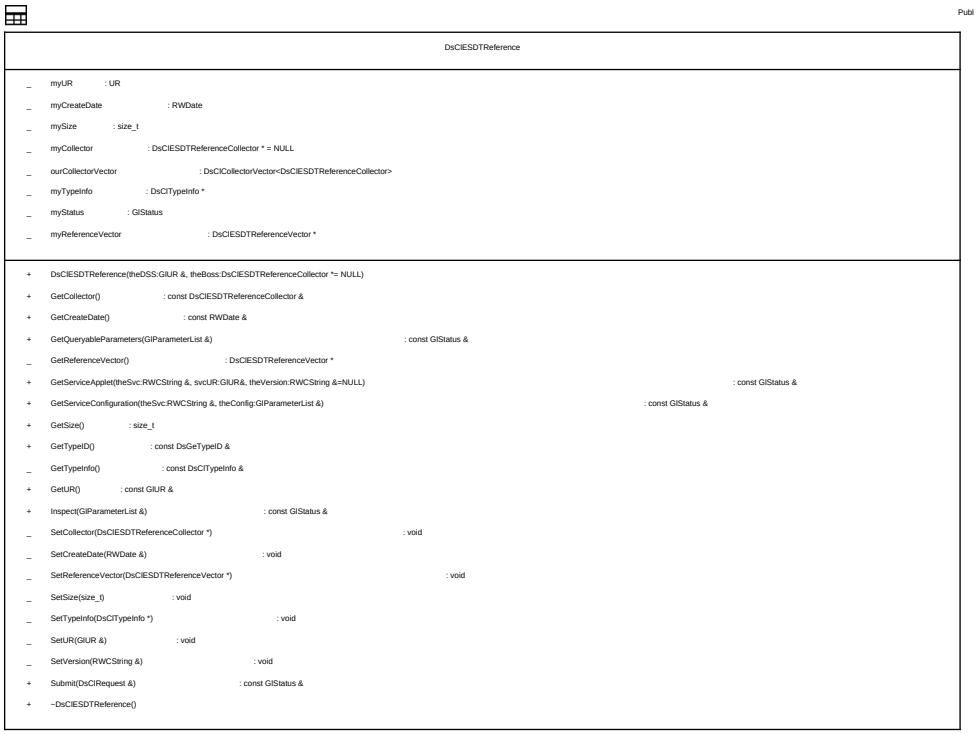


Figure 4.3-3. DsClESDT Object Model Diagram

DsClQuery

_	myStatus : GlStatus
_	QueryType : DsTQueryType = Inventory
_	AttributesToReturn : GlParameterList = ("all", char, "*")
_	Constraints : GlParameterList
_	MaxHits : int
_	myLocalCallback : GlCallback
+	DsClQuery()
+	DsClQuery(GlParameterList &)
+	~DsClQuery()
+	SetQueryType(DsTQueryType) : GlStatus &
+	GetQueryType() : DsTQueryType
+	SetAttributesToReturn(GlParameterList &) : GlStatus &
+	GetAttributesToReturn(GlParameterList &) : GlStatus &
+	SetConstraints(GlParameterList &) : GlStatus &
+	GetConstraints(GlParameterList &) : GlStatus &
+	SetMaxHits(int) : GlStatus &
+	GetMaxHits(int &) : GlStatus &
+	SetCallback(GlCallback &) : GlStatus &
+	GetCallback(GlCallback &) : GlStatus &
+	ConvertFromSQL(RWCString &) : GlStatus &
+	ConvertToSQL(RWCString &) : RWCString &
±	ConvertToCommand(DsClRequest &) : DsClRequest &

DsClSubscription

± $	ourCollectorvector : DsClSubscriptionCollectorVector &
±	myCollector : DsClSubscriptionCollector&
±	mySubmittedFlag : RWBoolean = RWTrue
±	myDescription : RWCString
±	myUserInfo : DsClClient&
±	myExpirationDate : RWDate
±	myAction : DsClAction
±	myDurationType : enum DsEClSubscriptionType = {ONCE, OUTSTANDING}
+	DsClSubscription()
+	DsClSubscription(submittedflag, DsClSubscriptionCollector&, Stream)
+	DsClSubscription(userinfo, Advertisement&, DsClSubscriptionCollector&)
+	GetAction(DsClAction &) : void
+	GetDescriptoin() : RWCString
+	GetDurationtype() : DsEClSubscriptionType
+	GetExpirationdate() : RWDate
_	GetSubmittedflag() : RWBoolean
+	GetUserinfo(GLClient&)
+	SetAction(DsClAction&) : void
+	SetDescription(RWCString) : void
+	SetDurationType(DsEClSubscriptionType) : void
+	SetExpirationDate(RWDate) : void
+	SetSubmittedFlag(RWBoolean) : void
+	Submit() : GlStatus&
+	Withdraw() : GlStatus&
+	~DsClSubscription()

Figure 4.3-4. DsClQuery_Subscription Object Model Diagram

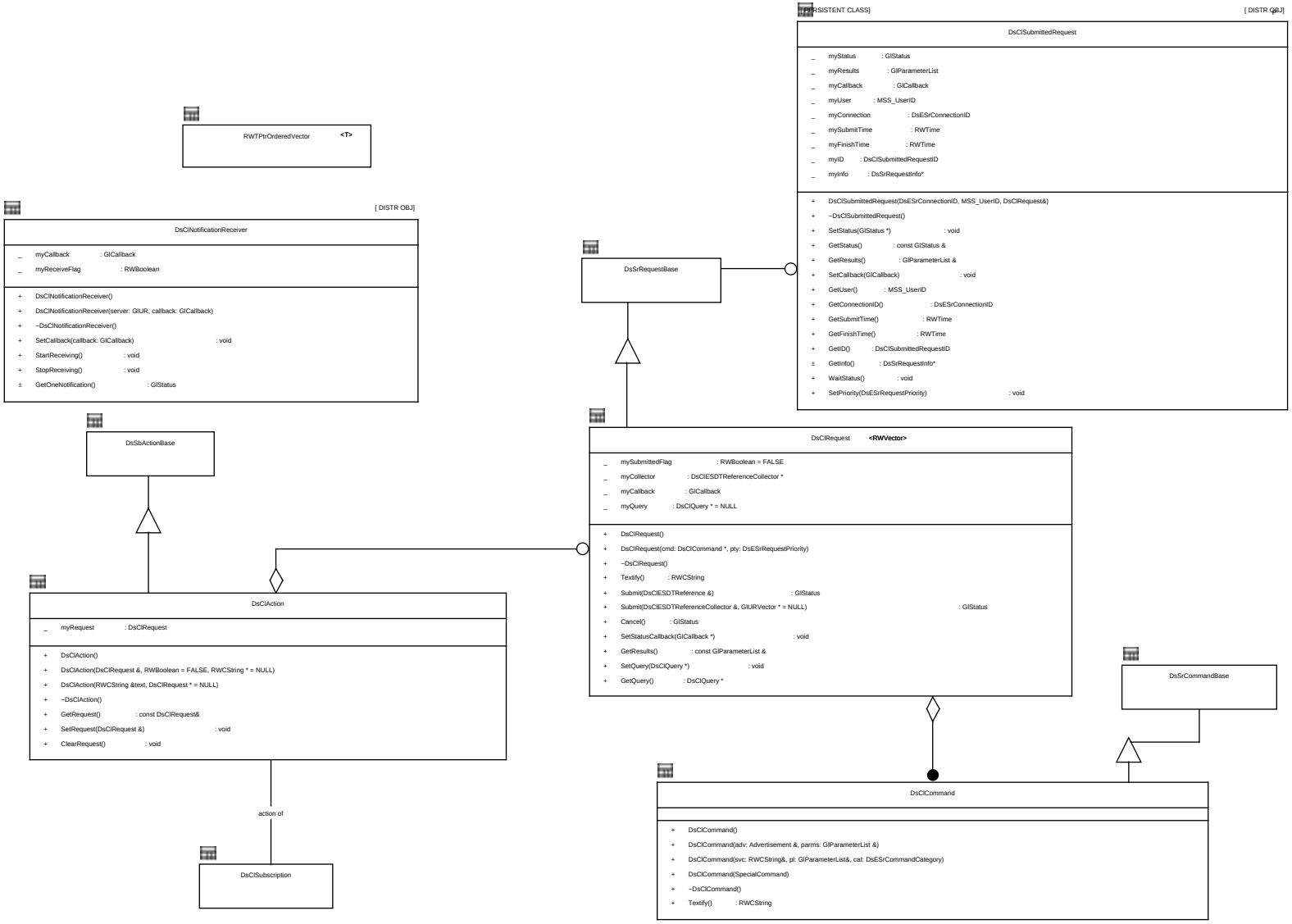


Figure 4.3-5. DsClRequest Object Model Diagram

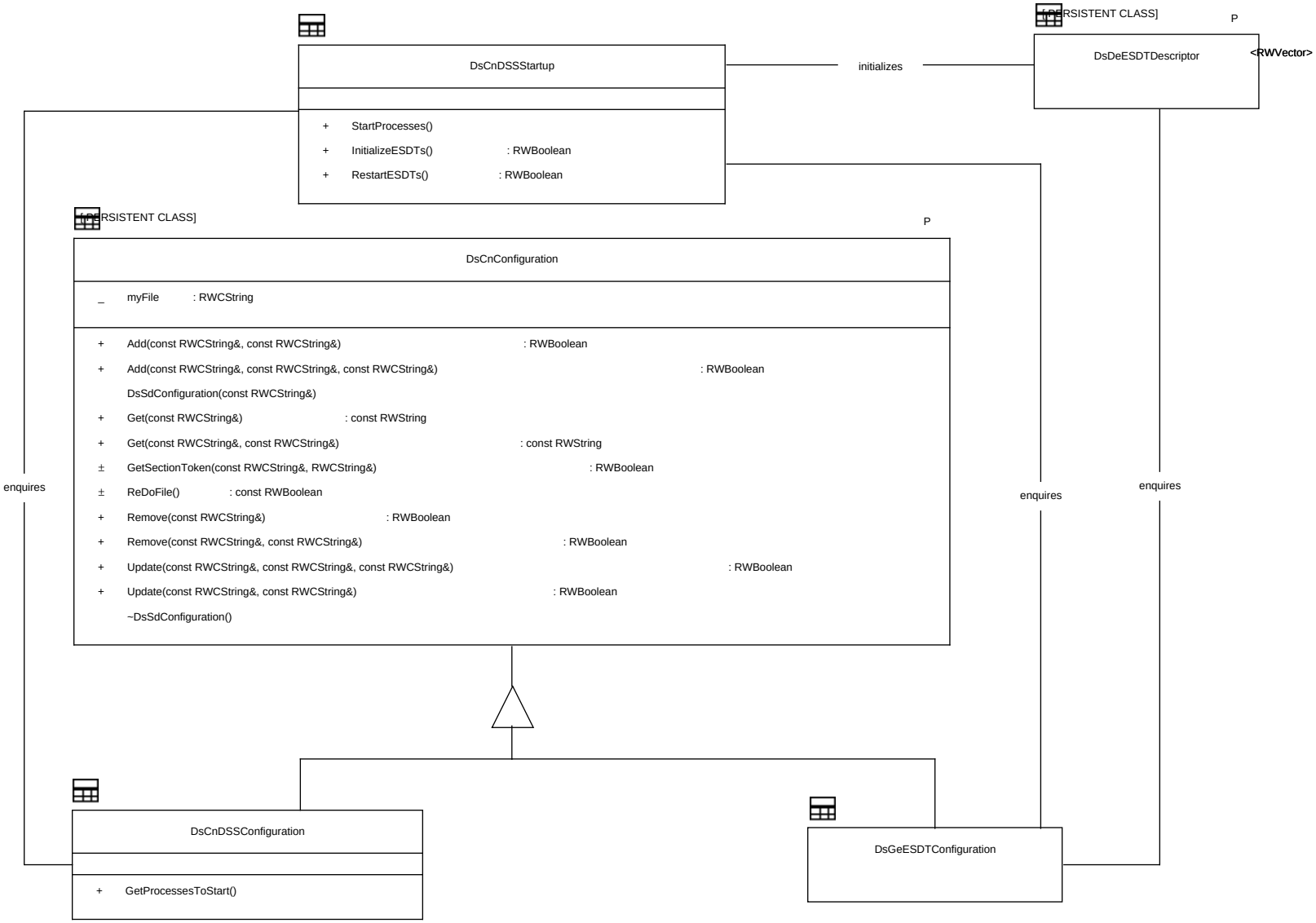


Figure 4.3-6. DsCnConfiguration Object Model Diagram

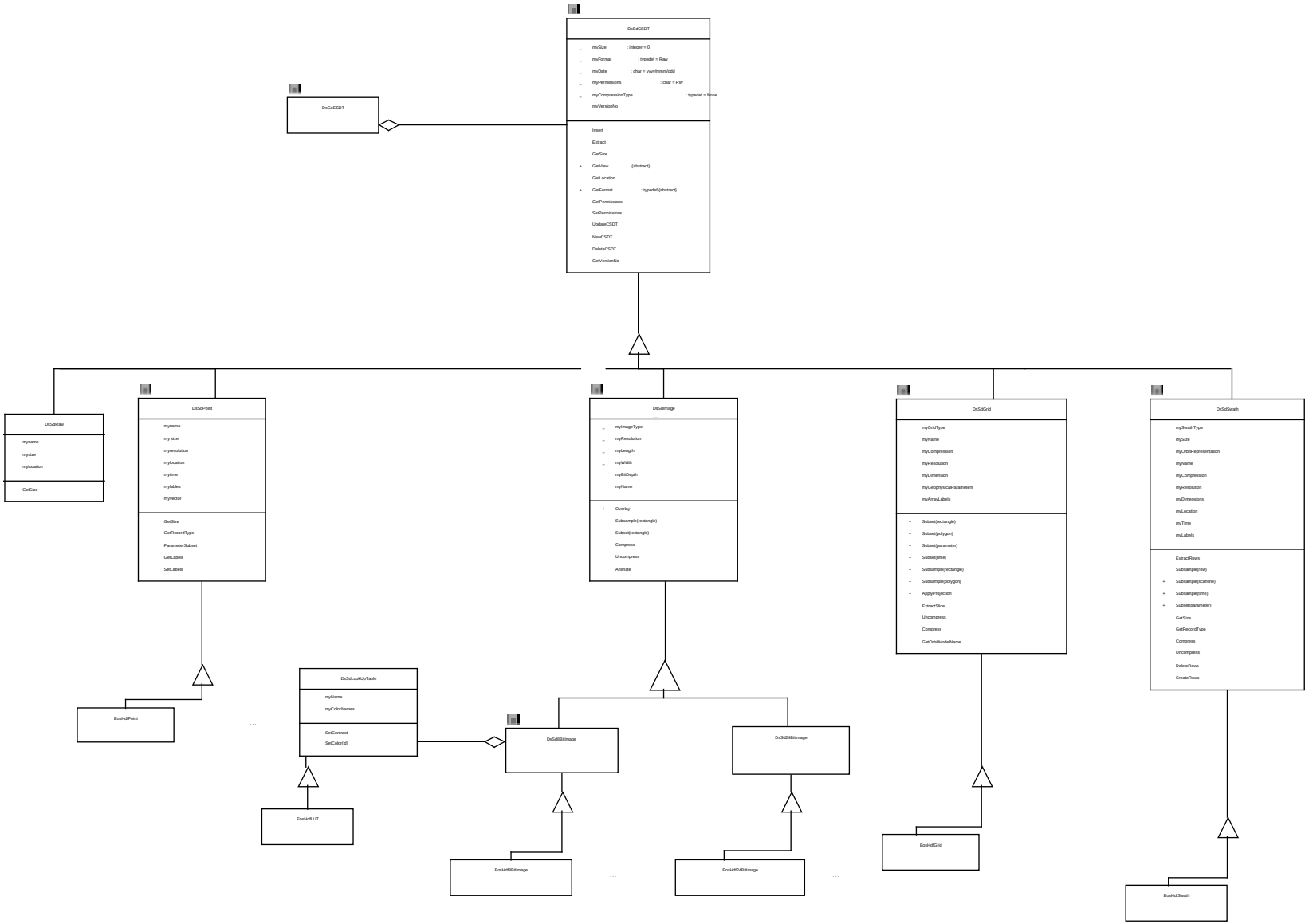


Figure 4.3-7. DsCsCSDT Object Model Diagram

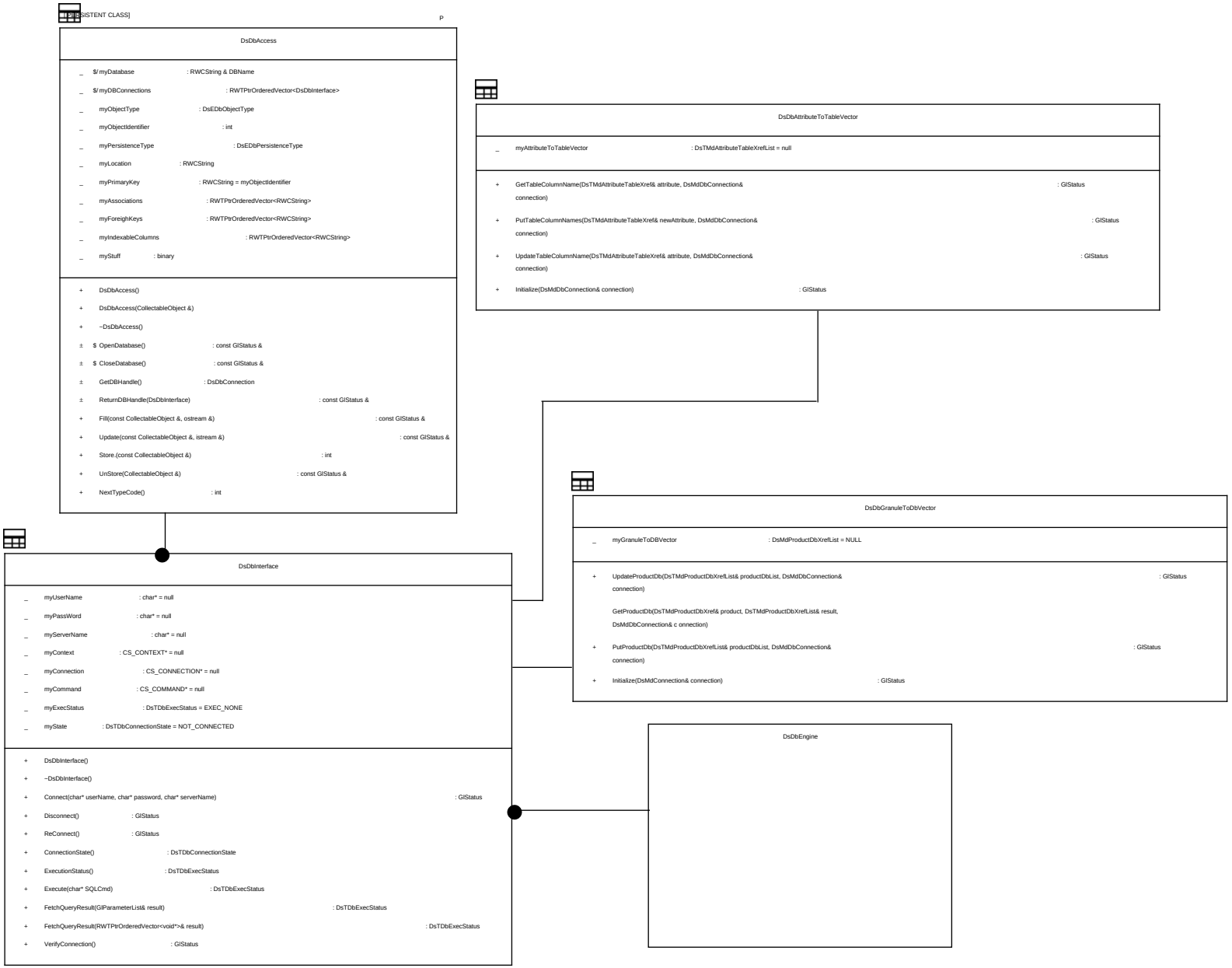


Figure 4.3-8. DsDbWrapper Object Model Diagram

This page intentionally left blank.

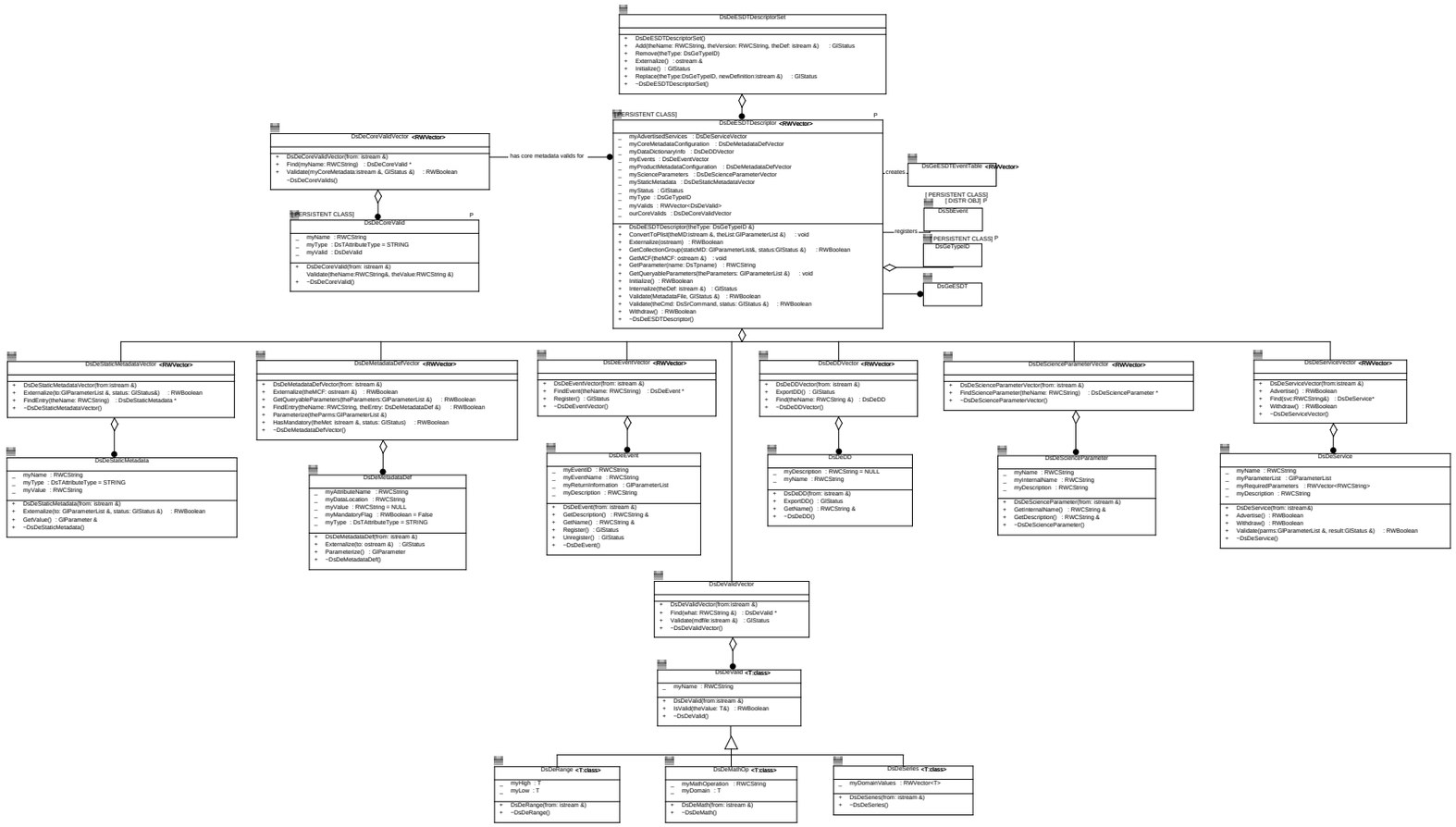


Figure 4.3-9. DsDeESDTDescriptor Object Model Diagram

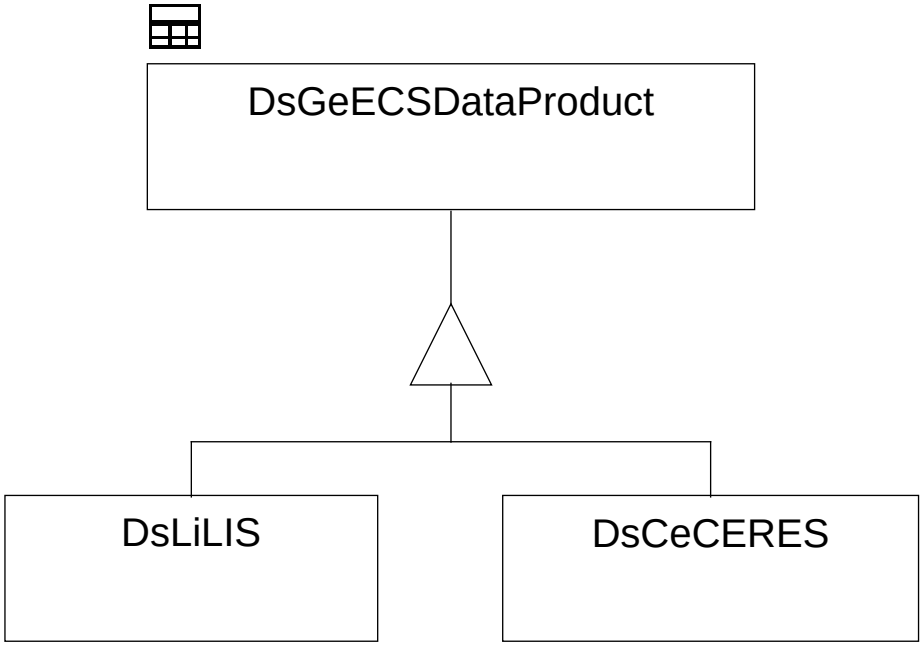


Figure 4.3-10. DsGeECSDataProductsRelA Object Model Diagram

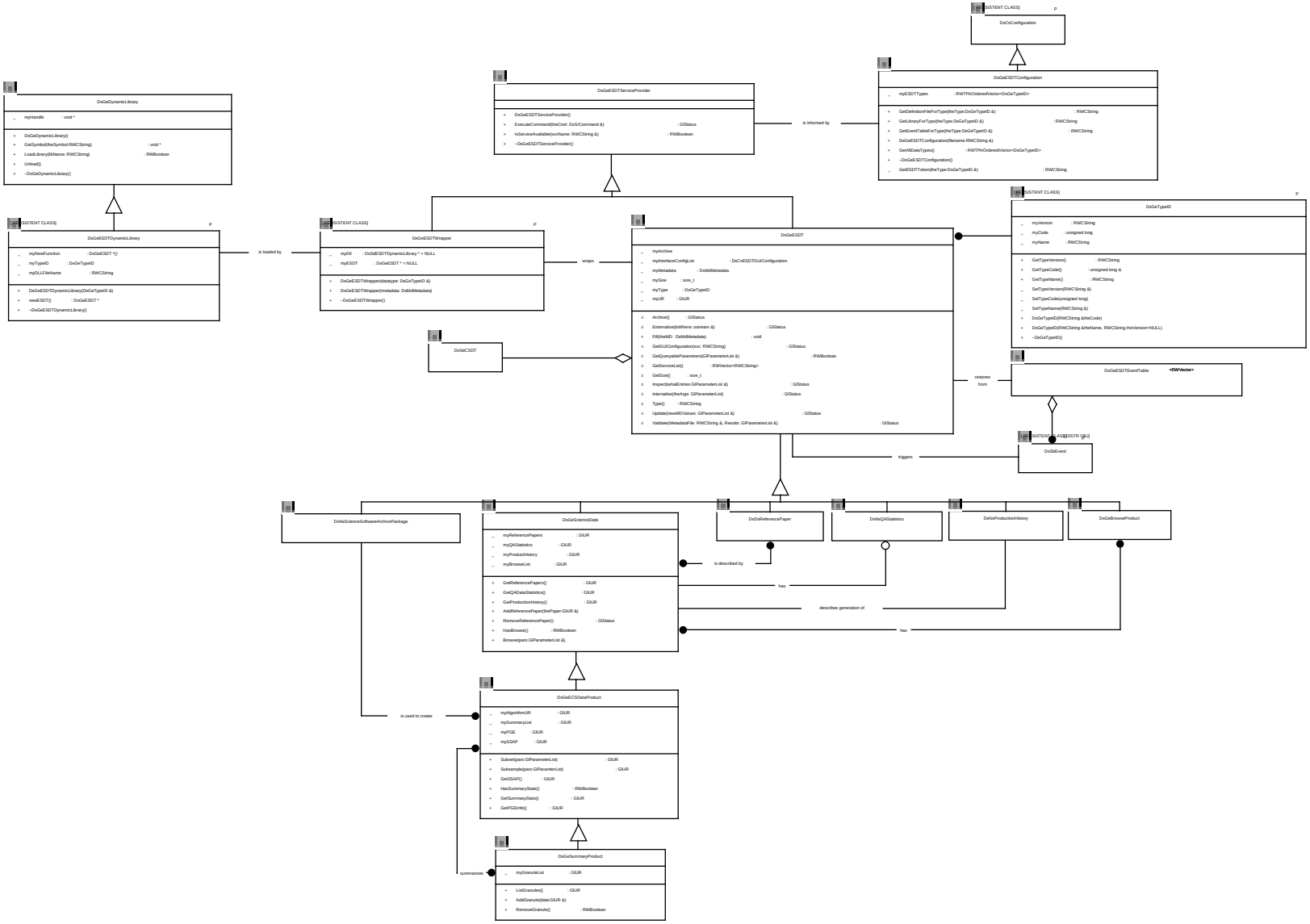


Figure 4.3-11. DsGeGeneralESDT Object Model Diagram

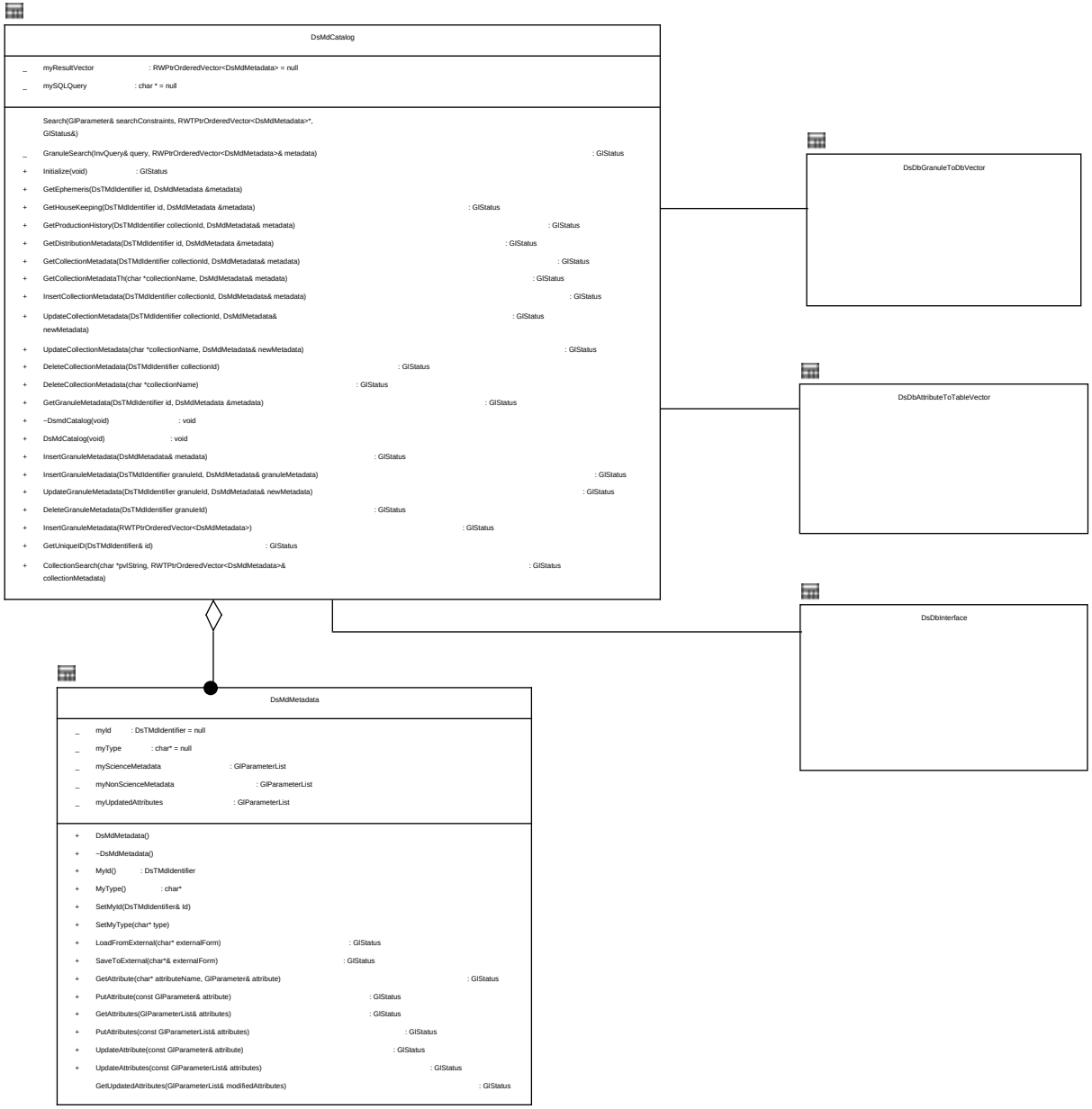


Figure 4.3-12. DsMdMetadata Object Model Diagram

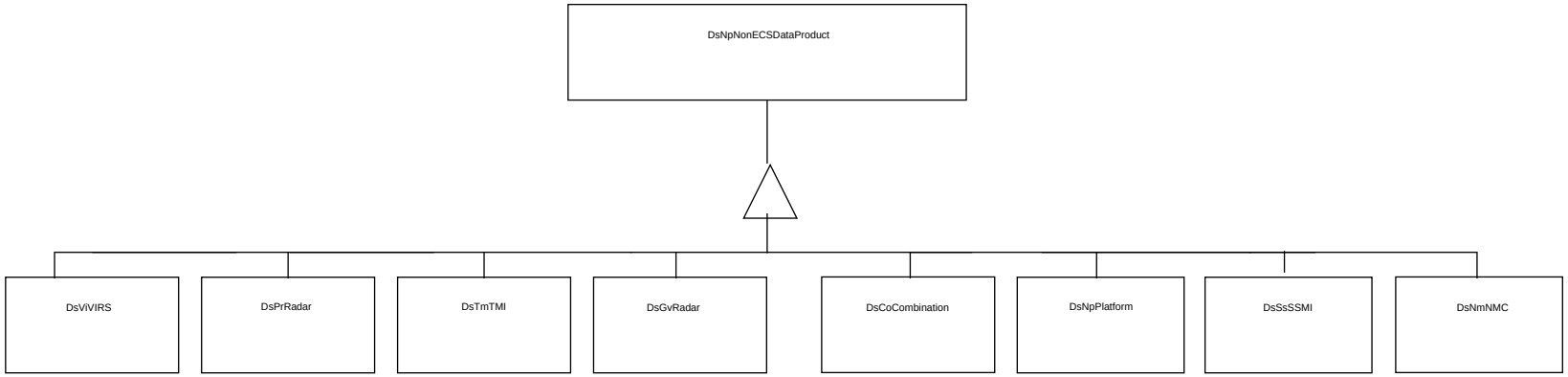


Figure 4.3-13. DsNpNonECSDataProduct Object Model Diagram

Figure 4.3-14. DsNpNonECSScienceData Object Model Diagram

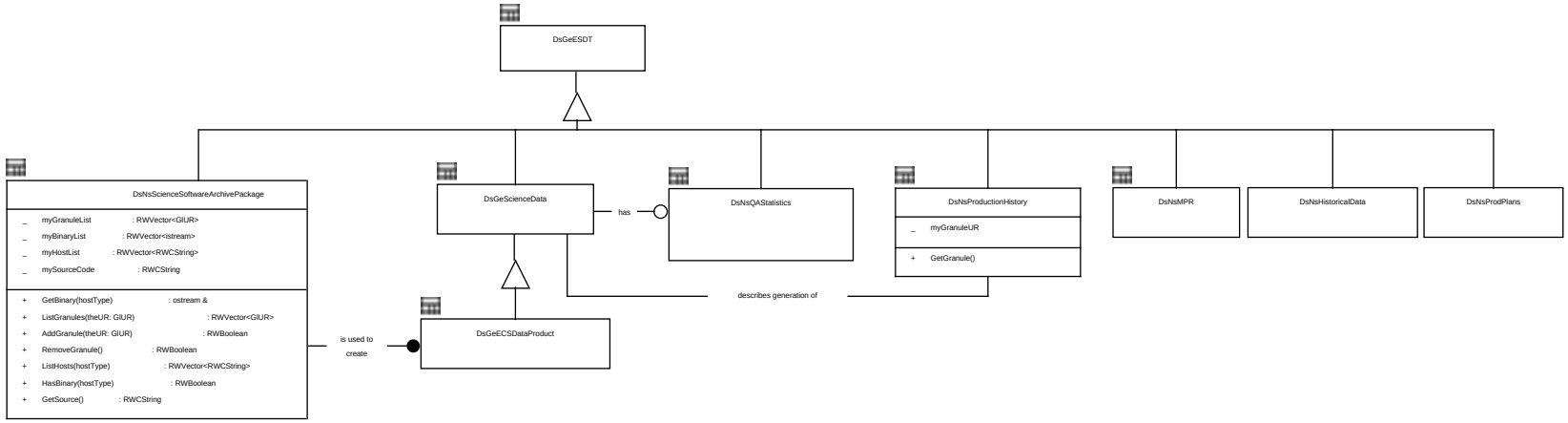


Figure 4.3-15. DsNsNonScienceESDT Object Model Diagram

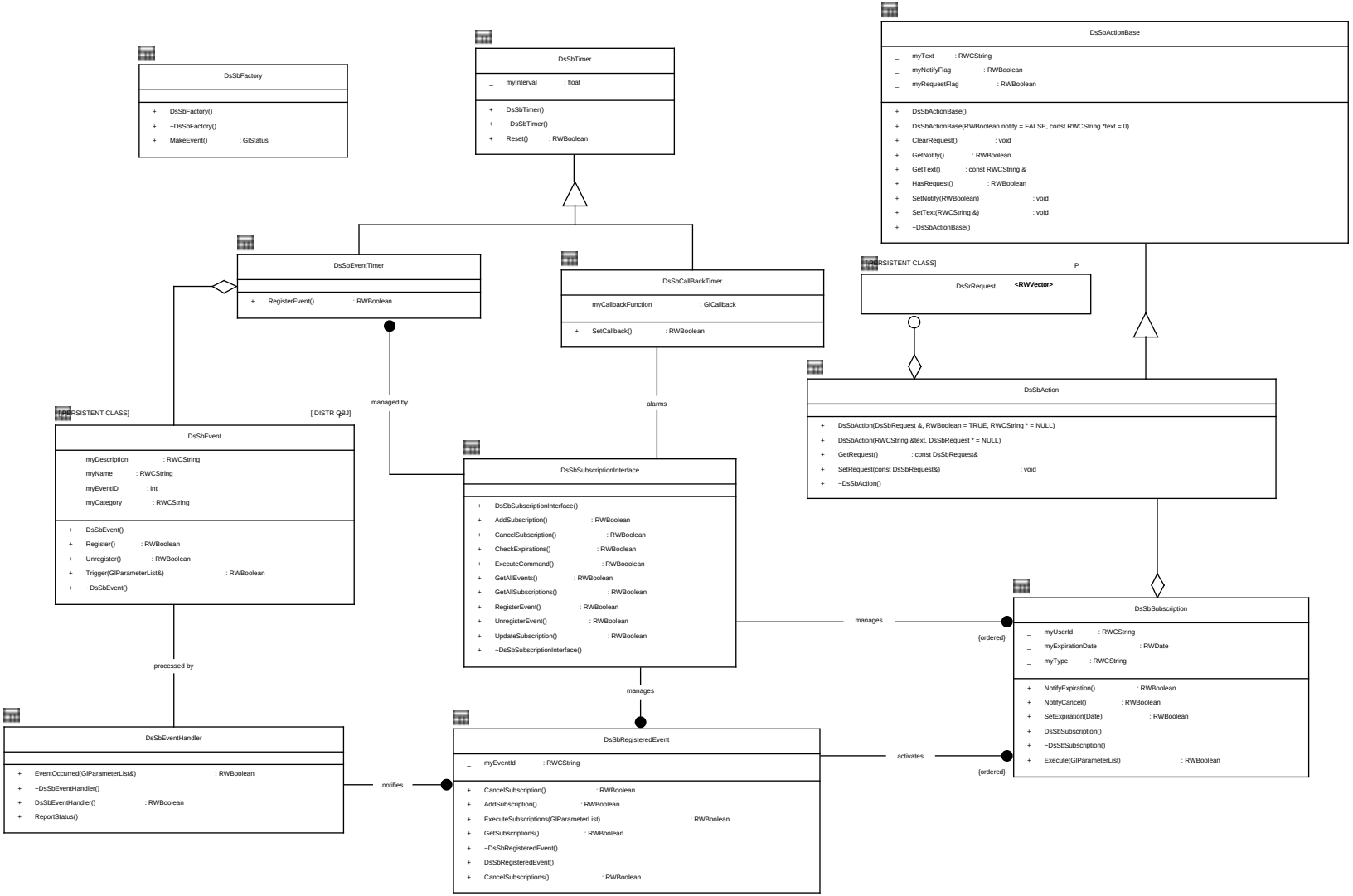


Figure 4.3-16. DsSbSubscription Object Model Diagram

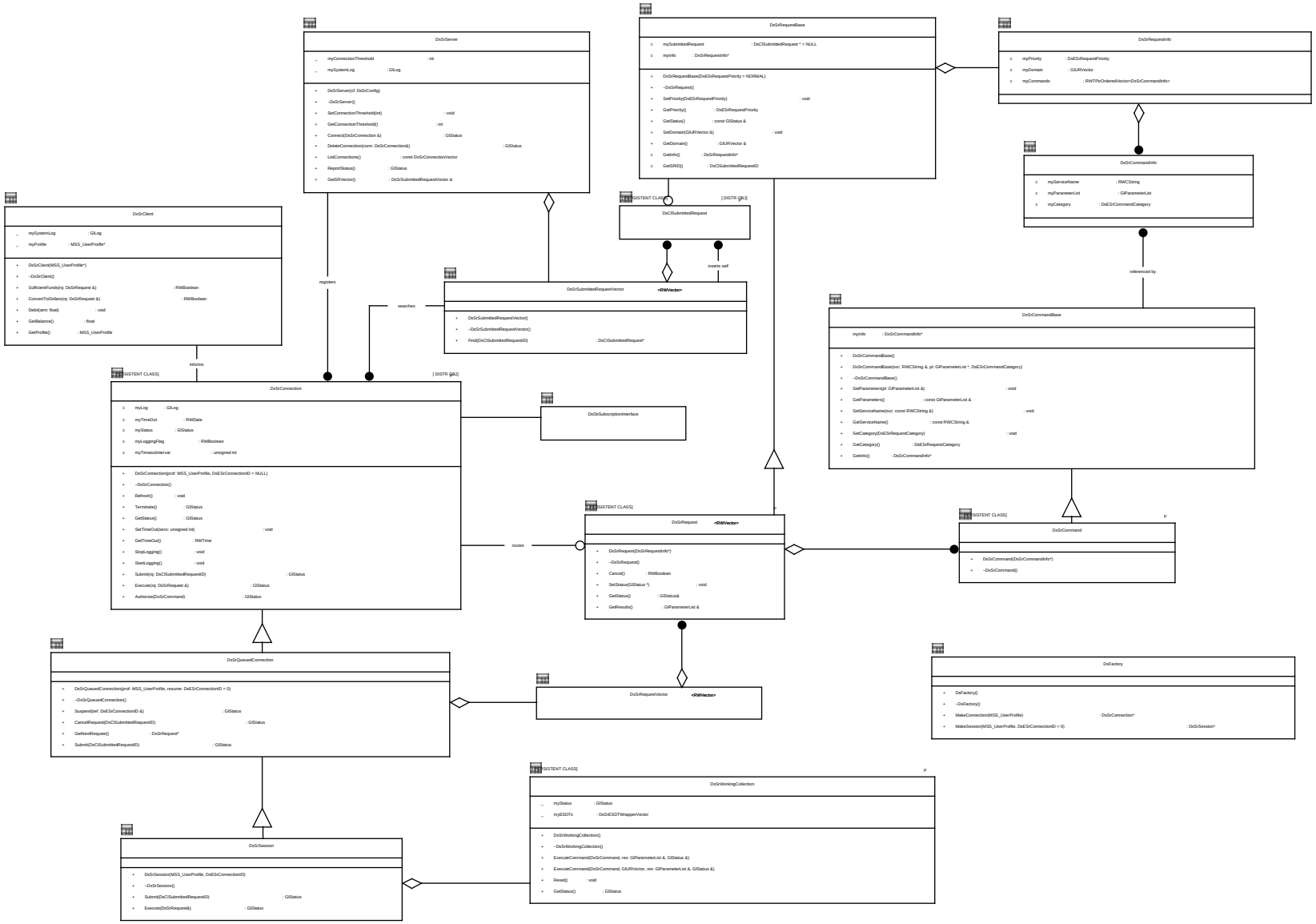


Figure 4.3-17. DsSrScienceServer Object Model Diagram

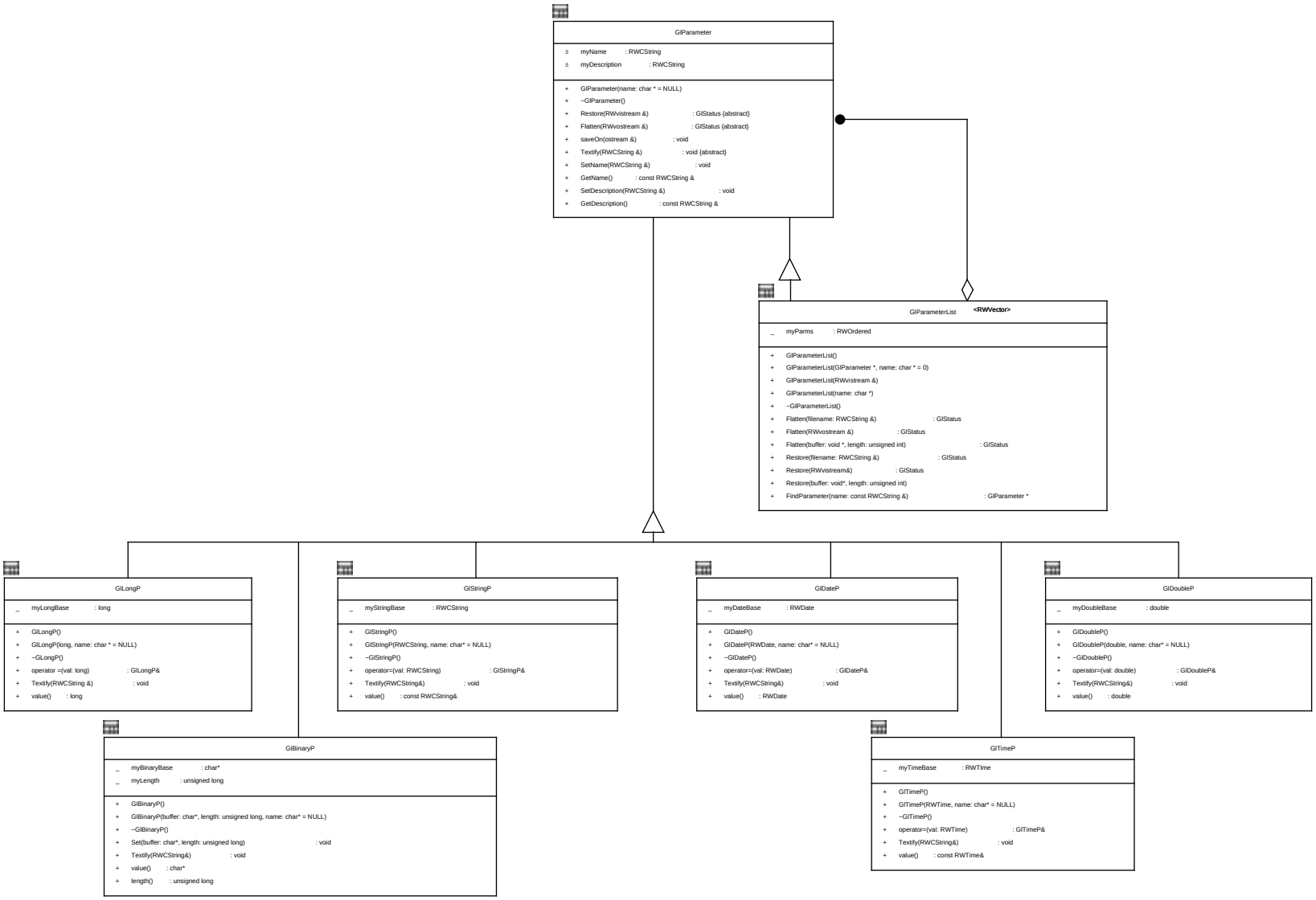


Figure 4.3-18. GIParameter Object Model Diagram

4.3.1 DsAdDataTypeCollector Class

Parent Class: Not Applicable
Public: No Distributed Object: Yes
Purpose and Description:
Provides administration/operations and interface to collectively handle descriptors (i.e., from the database).

Attributes:

None

Operations:

DsAdDataTypeCollector -
Arguments: GlUR&
Return Type: Void
Privilege: Public

ListTypes - Lists all data types (i.e., in the database).
Arguments:
Return Type: GlParameterList
Privilege: Public

NewType
Arguments: ur: GlUR&, name: RWCString&, vsn: RWCString&, descripfile: istream&
Return Type: Void
Privilege: Public

~DsAdDataTypeCollector
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsAdDataTypeCollector class has associations with the following classes:
None

4.3.2 DsAdDescriptor Class

Parent Class: DsClDescriptor
Public: No Distributed Object: No
Purpose and Description:
Provides more functionality than the client descriptor with the ability to update descriptors as an administrative interface to them.

Attributes:

All Attributes inherited from parent class

Operations:

DsAdDescriptor - Constructs an operations descriptor for a particular dataserver and stream.
Arguments: type: RWCString&, vsn: RWCString&, client: GlClient&, ds: GlUR&
Return Type: Void
Privilege: Public

DsAdDescriptor
Arguments: type: RWCString&, vsn: RWCString&, client: GlClient&, ds: GlUR&, strm: iostream&
Return Type: Void
Privilege: Public

Update - Update the descriptors given a list of parameters.
Arguments: GlParameterList&
Return Type: Void
Privilege: Public

~DsAdDescriptor
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsAdDescriptor class has associations with the following classes:
None

4.3.3 DsAdLog Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
Provides administrative logging functions as defined by the MSS object. This allows the operations staff to categorize messages, reports, of errors, and view resources used/ allocated by the client.

Attributes:

None

Operations:

DsAdLog - Constructs DsAdLog class.
Arguments:
Return Type: Void
Privilege: Public

~DsAdLog
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsAdLog class has associations with the following classes:
None

4.3.4 DsAdRequestInterface Class

Parent Class: Not Applicable
Public: No Distributed Object: Yes
Purpose and Description:
Provides administration/operations staff an interface to look at active and queued requests as well as status information of requests.

Attributes:

None

Operations:

DsAdRequestInterface - Constructs the administration/operations request interface given a particular dataserver and client.
Arguments: GlUR&, GlClient&
Return Type: Void
Privilege: Public

ListAllQueuedRequests - Accesses client queued requests classes in order to view, get stats, and delete requests.
Arguments:
Return Type: GlStatus&
Privilege: Public

ListAllRequests -
Arguments:
Return Type: GlStatus&
Privilege: Public

SetQueueSize - Allows the operations staff to set threshold for the number of service requests to be queued for processing.
Arguments: threshold: int
Return Type: RWBoolean
Privilege: Public

~DsAdRequestInterface
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsAdRequestInterface class has associations with the following classes:
None

4.3.5 DsCeCERES Class

Parent Class: DsGeECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class represents products generated from the TRMM CERES Instrument data.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsCeCERES class has associations with the following classes:
None

4.3.6 DsClAction Class

Parent Class: DsSbActionBase
Public: No Distributed Object: No
Purpose and Description:
A client interface object that represents the components of the action to be performed when a subscription is triggered. The possibilities are that the client will receive a notification (including all parameters that are returned by the object that triggers the subscription and an optional piece of client-specified text) and/or a request that will be executed. The client is required to specify an action for each subscription.

Attributes:

myRequest - The request that is currently associated with this action.
Data Type: DsClRequest
Privilege: Private
Default Value:

Operations:

ClearRequest - Used to clear any request that has been set for the action.
Arguments:
Return Type: void
Privilege: Public

DsClAction - Used to construct an action from a piece of text and, optionally, a request. The notification flag is set.
Arguments: RWCString &text, DsClRequest * = NULL
Return Type: Void
Privilege: Public

DsClAction - Used to construct an action from a request and, optionally, a value for the notification flag and a piece of text.
Arguments: DsClRequest &, RWBoolean = FALSE, RWCString * = NULL
Return Type: Void
Privilege: Public

DsClAction - Constructs a default action.
Arguments:
Return Type: Void
Privilege: Public

GetRequest - Returns the request currently set for the action.
Arguments:
Return Type: const DsClRequest&
Privilege: Public

SetRequest - Sets the request to be executed when the subscription fires.
Arguments: DsClRequest &
Return Type: void
Privilege: Public

~DsClAction - Used to destroy an action.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClAction class has associations with the following classes:
Class: DsClSubscription actionof

4.3.7 DsClCollector Class

Parent Class: Not Applicable

Public: No Distributed Object: Yes

Purpose and Description:

This is an abstract, distributed base class for the Collector distributed objects. This class is the client portion of an IDL definition (the corresponding server portion is Connection). This division of function relieves all classes in the DS public class library from having to be knowledgeable of the dataserver object realm. This class inherits from the Rogue Wave RWTPtrOrderedVector and provides all the normal vector behaviors to the specialized classes. There are no attributes for this object.

Attributes:

None

Operations:

DsClCollector

Arguments: GlUR &dataserver, MSS_UserProfile &

Return Type: Void

Privilege: Public

GetConnectionID - This public operation allows client software to retrieve the dataserver-assigned session ID so that users can turn off their terminals during long-running request processing, then log on later and reconnect to the session.

Arguments:

Return Type: DsESrConnectionID

Privilege: Public

SubmitToServer - This protected operation passes a request from a DsClRequest to the dataserver after the DsClRequest has established the DsClSubmittedRequest to handle the distributed callback. This operation is fully implemented on the server side (i.e. in either DsSdConnection or the specialized connection object DsSdSession). All return values are handled by the DsClRequest object, which contains imbedded return parameters for this purpose.

Arguments: DsClRequest&

Return Type: const GlStatus &

Privilege: Protected

~DsClCollector
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClCollector class has associations with the following classes:
Class: DsClCollectorVector collectedby

4.3.8 DsClCollectorVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This private class is maintained in static memory by DsCl items (DsClSubscription and DsClESDTReference). Those items contain a DsClCollectorVector. This class supports the "hidden construction" of collector objects (and hence, connections to a dataserver). It is a RWTPtrOrderedVector, which is parameterized so that the item which is maintaining it can decide what type of collector to store and track. This class represents a table of available DsCl<item>Collectors which were created by item-level DsCl public classes (i.e., DsClSubscription or DsClESDTReference). This table will contain a pointer to one DsCl<item>Collector per dataserver. All item-level objects created in the client space independently of a client-created DsCl<item>Collector will be assigned to the default DsCl<item>Collector as defined by the entry in the DsClCollectorVector which corresponds to the dataserver for which the client is creating the independent item. (phew!)

Attributes:

None

Operations:

DsClCollectorVector - This is the default constructor, for initial startup.
Arguments:
Return Type: Void
Privilege: Public

GetCollector - This operation finds the existing collector for the given dataserver (if any) or creates one if none exists. It returns the reference to the chosen collector. Because the

class is a template, the return value will match (be the right kind of collector for) the object which is calling this operation.
Arguments: const GlUr &dataserver
Return Type: T&
Privilege: Public

~DsClCollectorVector - The destructor should presumably be called only when the vector is empty, but it might be a good idea to check for any collectors here and close them down.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClCollectorVector class has associations with the following classes:
Class: DsClCollector collectedby

4.3.9 DsClCommand Class

Parent Class: DsSrCommandBase
Public: No Distributed Object: No
Purpose and Description:
A specialization of the DsCommand class for client interfaces. Adds constructors that ease building of commands based on advertisements, or special direct commands that are "built-in" to the data server and do not correspond to advertisements.

Attributes:

All Attributes inherited from parent class

Operations:

DsClCommand - Used to construct "special" commands, i.e. commands that are not advertised. For example, the command to reset the working collection.
Arguments: SpecialCommand
Return Type: Void
Privilege: Public

DsClCommand - Used to construct a command from an advertisement. The GlParameterList gives the parameters required by the chosen command.

Arguments: adv: Advertisement &, parms: GlParameterList &
Return Type: Void
Privilege: Public

DsClCommand - Constructs an empty command.
Arguments:
Return Type: Void
Privilege: Public

DsClCommand - Used to construct a command from its basic parts: service name, parameters, and category.
Arguments: svc: RWCString&, pl: GlParameterList&, cat: DsESrCommandCategory
Return Type: Void
Privilege: Public

Textify - Used to convert a command into a human-readable format.
Arguments:
Return Type: RWCString
Privilege: Public

~DsClCommand - Used to destroy a command.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClCommand class has associations with the following classes:
DsClRequest (Aggregation)

4.3.10 DsClDescriptor Class

Parent Class: Not Applicable
Public: No Distributed Object: Yes
Purpose and Description:
This object provides access to services pertaining to the definition of a specific data type. This includes access to metadata configuration information and queriable parameters. This object validates metadata.

Attributes:

myStatus - The status attribute captures status information about activities that are performed on this instance.
Data Type: GlStatus
Privilege: Private
Default Value:

myTypeID - Reference to an object that uniquely identifies the specific type of this descriptor.
Data Type: DsGeTypeID *
Privilege: Private
Default Value:

ourCollectorVector - Reference to a static vector of collectors. Each collector is connected to a data server. Upon construction, the descriptor must check to see if there is a collector connected to its data server. If so, the descriptor add itself to that collectors list of items. If not, the descriptor creates a collector that talks to the desired data server and adds it to the static vector collector set.
Data Type: RWVector *
Privilege: Private
Default Value:

Operations:

DsClDescriptor - Constructor for the descriptor. The arguments are references to a client object, a dataserver, and a typeID object indicating the type of the descriptor to create.
Arguments: GlClient &, GlUR &,DsGeTypeID &
Return Type: Void
Privilege: Private

GetCollector - Returns a pointer to the DescriptorCollector that contains this descriptor.
Arguments:
Return Type: DsClDescriptorCollector*
Privilege: Private

GetMCF - Returns a stream containing the metadata configuration file (MCF) data.
Arguments: ostream &
Return Type: GlStatus
Privilege: Public

GetQueryableParameters - The service takes a reference to an empty GlParameterList and fills that list in with the queriable parameters and their associated types for this data type.
Arguments: GlParameterList &
Return Type: RWBoolean
Privilege: Public

GetTypeID - Returns a reference to the DsSdTypeID object which contains information to uniquely identify the type of this descriptor.
Arguments:
Return Type: DsGeTypeID &
Privilege: Public

SetTypeID - This is a private function to set the typeid for this object.
Arguments: DsGeTypeID &
Return Type: Void
Privilege: Public

Validate - This service takes a stream which contains metadata and validates this metadata. The GlStatus object that is returns provides an indication of whether the validation was successful.
Arguments: istream &metadata
Return Type: GlStatus
Privilege: Public

~DsClDescriptor - Destructor for the descriptor. If this is the last descriptor of the hidden collector to be destroyed, then the collector is also destroyed.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClDescriptor class has associations with the following classes:
None

4.3.11 DsClESDTReference Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:

This object is a reference to an ESDT that is within a DataServer's holdings. This object provides services that are homogeneous for all ESDTs.

Attributes:

myCollector - A pointer to the collector that this reference is a member of. If this pointer is null, then this reference is a member of one of the collectors in the static collector vector.
Data Type: DsClESDTReferenceCollector *
Privilege: Private
Default Value: NULL

myCreateDate - The creation date of the ESDT that this reference represents.
Data Type: RWDate
Privilege: Private
Default Value:

myReferenceVector
Data Type: DsClESDTReferenceVector *
Privilege: Private
Default Value:

mySize - This number of MB that the underlying granule would occupy if the ESDT that this reference points to was externalized.
Data Type: size_t
Privilege: Private
Default Value:

myStatus
Data Type: GlStatus
Privilege: Private
Default Value:

myTypeInfo - A pointer to an object which contains information related to all ESDT references of the same type.
Data Type: DsClTypeInfo *
Privilege: Private
Default Value:

myUR - The unique reference that is assigned to this ESDT reference.
Data Type: UR
Privilege: Private
Default Value:

ourCollectorVector - A static vector of ESDT collectors. Each ESDT reference is collected by an ESDT collector. The collector vector is used to allow creation of an ESDT reference without first establishing a collector.
Data Type: DsClCollectorVector<DsClESDTReferenceCollector>
Privilege: Private
Default Value:

Operations:

DsClESDTReference
Arguments: theDSS:GlUR &, theBoss:DsClESDTReferenceCollector *= NULL
Return Type: Void
Privilege: Public

GetCollector
Arguments:
Return Type: const DsClESDTReferenceCollector &
Privilege: Public

GetCreateDate - Returns the date that the underlying ESDT granule was created.
Arguments:
Return Type: const RWDate &
Privilege: Public

GetQueryableParameters - This service is used to obtain the queryable parameters for this ESDT type. The queryable parameters will be the same for all ESDT references of the same type. For example, all CER03s have the same queryable parameters. The input argument is a reference to an empty parameter list. The return value is a boolean indicating success or failure of the request. If the request was successful, the parameter list will have the names and types (but no values) of all queryable parameters for this TYPE.
Arguments: GlParameterList &
Return Type: const GlStatus &
Privilege: Public

GetReferenceVector
Arguments:
Return Type: DsClESDTReferenceVector *
Privilege: Private

GetServiceApplet
Arguments: theSvc:RWCString &, svcUR:GlUR&, theVersion:RWCString &=NULL
Return Type: const GlStatus &
Privilege: Public

GetServiceConfiguration
Arguments: theSvc:RWCString &, theConfig:GlParameterList &
Return Type: const GlStatus &
Privilege: Public

GetSize - Returns the size of data granule associated with this ESDT Reference. This is the size of the granule if externalized.
Arguments:
Return Type: size_t
Privilege: Public

GetTypeID - Returns a reference to an object that contains all of the information that uniquely identifies this type.
Arguments:
Return Type: const DsGeTypeID &
Privilege: Public

GetTypeInfo
Arguments:
Return Type: const DsClTypeInfo &
Privilege: Private

GetUR - Returns the UR that is associated with this ESDT reference.
Arguments:
Return Type: const GlUR &
Privilege: Public

Inspect - This service returns values for given parameters. The parameter list has the names of the parameters for which values are desired and this service returns the same parameter list with the values and associated types filled in. A boolean is also returned to indicate success or failure of the service request.
Arguments: GlParameterList &
Return Type: const GlStatus &
Privilege: Public
PDL:
DsClESDTReference::Inspect(GlParameterList &theParms)

This operation gets the values for the metadata attributes specified in the given GlParameterList. This information is actually obtained from the real ESDT in the corresponding DsSrWorkingCollection on the Data Server Subsystem. A request is created and submitted to the DSS through the DsClESDTReferenceCollector for this object.

```
SEQUENCE
  Create DsClCommand with arguments ("INSPECT", theParms, ESDT)
  Create DsClRequest with arguments (Command, NORMAL)
  SetCallback of DsClRequest to ReturnResults
  Submit the request myCollector
  Retrieve results from request
  Return results to client
END SEQUENCE
```

SetCollector - This is a private member function used to set the value of the myCollector attribute.
Arguments: DsClESDTReferenceCollector *
Return Type: void
Privilege: Private

SetCreateDate - This is a private service used to set the myCreateDate attribute.
Arguments: RWDate &
Return Type: void
Privilege: Private

SetReferenceVector - This is a private service used to set the myReferenceVector attribute.
Arguments: DsClESDTReferenceVector *
Return Type: void
Privilege: Private

SetSize - This is a private member function used to set the mySize attribute.
Arguments: size_t
Return Type: void
Privilege: Private

SetTypeInfo - This is a private service used to set the myTypeInfo attribute.
Arguments: DsClTypeInfo *
Return Type: void
Privilege: Private

SetUR - This is a private service used to set the myUR attribute.
Arguments: GlUR &
Return Type: void
Privilege: Private

SetVersion
Arguments: RWCString &

Return Type: void
Privilege: Private

Submit
Arguments: DsClRequest &
Return Type: const GlStatus &
Privilege: Public

~DsClESDTReference - Destructor for the ESDT Reference. If this descriptor is a member of a collector in the static collector vector and this is the last descriptor to be destroyed, then its collector is also destroyed.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClESDTReference class has associations with the following classes:
None

4.3.12 DsClESDTReferenceCollector Class

Parent Class: DsClCollector
Public: No Distributed Object: Yes
Purpose and Description:
This public, distributed class is a specialization of the Collector class which handles DsClESDTReferences. This class is much more complex than the base class. This class provides, in addition to the normal set operations for ESDTReferences, the ability to handle requests, working-collection synchronization, and sessions. It also contains private operations to hand the ESDTReference-level actions to the dataserver.

Attributes:

mySearchCallback - This attribute is for specifying the local callback for a query.
Data Type: GlCallback
Privilege: Private
Default Value:

myStatus - This attribute allows the object to maintain information on current status.
Data Type: GlStatus

Privilege: Private
Default Value:

Operations:

AddESDTReference - This operation is used to instantiate a DsClESDTReference when the (empty) object has already been allocated. For example, if the client creates a DsClESDTReference with a UR (which creates a DsClESDTReferenceCollector), this will turn into a request for the given UR. However, once the ESDT has been found and instantiated on the server side, and UpdateState will make the client side collection "match", the DsClESDTReference object that the client created must be used to store the reference, as the client software only knows about that actual memory address. So the UpdateState method will use a "replace" command to "fill in" the existing object which the client software has created. This is also useful for subsetting, when a DsClESDTReference already exists, but the metadata which describes it may change (become a smaller set) as a result of the subsetting operation. The same DsClESDTReference object should still be used, even though its contents have changed.
Arguments: const DsClESDTReference *ERef
Return Type: Void
Privilege: Public

BuildRequestVector - This private operation allows the DsClESDTReferenceCollector object to reconstruct the state of a session in the case that a user has "logged off" during a long-running request, and reconnects to the session to find the results. This operation reestablishes the client-side list of submitted requests (DsClSubmittedRequests) to check the status of the actual requests on the server side.
Arguments: GlURVector &
Return Type: const GlStatus &
Privilege: Protected

CreateESDTReference - This operation actually creates a DsClESDTReference object. It is used by the UpdateState method when an "add" command is received (i.e., no DsClESDTReference currently exists for the given UR).
Arguments: GlUR &
Return Type: DsClESDTReference*
Privilege: Protected
PDL:DsClESDTReferenceCollector::CreateESDTReference(GlUR &ESDT)

This operation is invoked in either of the following two cases:

1) a user has created a DsClESDTReferenceCollector and invoked this method directly

2) a user has created a DsClESDTReferenceCollector and submitted a Search request

In either case, the metadata for the ESDT identified by the provided GlUR is retrieved from the dataserver, and added to both the working collection (on the server side), and to the DsClESDTReferenceCollector (on the client side).

This operation returns the pointer to a DsClESDTReference. It is the responsibility of the calling procedure to manage the DsClESDTReference from that point onward.

SEQUENCE
  Create a request to retrieve the ESDT identified by the provided GlUR
  Submit the request
  Get new state information as a result of the request
  Update the state of the client-side collection to match the server-side
      working collection
  Return the pointer to the new object
END SEQUENCE

DeleteESDTReference - This operation is used during in the UpdateState method to actually remove client DsClESDTReference objects when their corresponding server-side objects have been removed from the DsSrWorkingCollection.
Arguments: DsClESDTReference*
Return Type: RWBoolean
Privilege: Protected

DsClESDTReferenceCollector
Arguments: GlUR &dataserver, MSS_UserProfile &, DsTSessionID = NULL
Return Type: Void
Privilege: Public

GetRequestVector - This operation allows client software to get a reference to the set of requests submitted during the current session, for whatever local processing (iteration) desired.
Arguments:
Return Type: const DsClRequestVector &
Privilege: Public

GetRequests - This private operation allows the DsClESDTReferenceCollector object to get a list of all the requests which have been submitted for the current session. This feature is to support the situation where a user has logged out while a long request is processing, then makes a new connection to the same SessionID to get the results. The collector object will use this operation in the "rebuilding" of the session state.
Arguments: GlURVector &

Return Type: const GlStatus &
Privilege: Protected

GetSessionLog - This is not for Release A. This operation allows the client software to declare an output stream and have the dataserver send over the portion of the system log which pertains to this particular session. All file-related programming is expected to occur in the client software.
Arguments: ostream &outfile
Return Type: const GlStatus &
Privilege: Public

Reset - This operation removes all current state for the collection, i.e. removes all existing ESDTReferences.
Arguments:
Return Type: const GlStatus &
Privilege: Public

ResumeSession - This is not for Release A. This operation takes the UR of a suspended session and recreates the state of that session. This operation will not be available in Release A.
Arguments: GlUR &SuspendedSession
Return Type: const GlStatus &
Privilege: Public

Search - This operation takes a DsClQuery object, which has been defined and filled in by the client software, and creates a DsClRequest (and associated DsClSubmittedRequest) and submits the request to the dataserver. The return values are included in the DsClRequest object, as are whatever commands are necessary to update the client working collection (this collection) to match the working collection on the server. The DsClESDTReferenceCollector *callback* function, which gets called by DsClRequest upon completion, will take the results of the DsClRequest, invoke UpdateState to make the client-side working collection match the server side one, then get the DsClQuery pointer from the DsClRequest so that it can call the Query object's callback function.
Arguments: DsClQuery &
Return Type: const GlStatus &
Privilege: Public

SetDialogCallback - This operation allows the client software to declare a callback for handling dialog-type notifications. It is not expected to be used in Release A.
Arguments: GlCallback&
Return Type: void
Privilege: Public

SetSearchCallback - This operation allows the client software to set a local callback for a query. This operation is private to the Search method.

Arguments: GlCallback&
Return Type: void
Privilege: Protected

SetStatusCallback - This operation allows the client software to establish a callback routine that will be the local callback function for all requests to report status to.
Arguments: GlCallback&
Return Type: void
Privilege: Public

Submit - This operation invokes the Submit of the DsClRequest object which establishes a DsClSubmittedRequest to handle the callback stuff. The DsClRequest will call the Collector (base class) SubmitToServer to actually send itself (request) across the network.
Arguments: DsClRequest &
Return Type: const GlStatus &
Privilege: Public

SuspendSession - This is not for Release A. This operation stores the state of the current session for later resumption. The UR of the session may be created at this point or it may be assigned when the session is first created. This operation will not be available in Release A.
Arguments: GlUR &SuspendedSession
Return Type: const GlStatus &
Privilege: Public

UpdateState - This private operation is used to make the client-space representation of the working collection match that on the server side. It takes commands from the GlParameterList and executes them in turn. Because of the possibility of multiple simultaneous queries, this operation must be concurrency-safe. Commands take one of the following *logical* forms. remove (existing_UR) add (new_UR, type (parameters), date, size) replace (existing_UR, new_type (parameters), date, size). If the ESDTReferenceVector (by type) is retained, the logic is as follows. add: if type doesn't exist, create type insert new_UR remove: remove existing_UR if last UR in type, remove type replace: if new_type doesn't exist, create type set parameters of existing_UR to new values if last UR in old_type, remove old_type
Arguments: GlParameterList &
Return Type: const GlStatus &
Privilege: Protected
PDL:DsClESDTReferenceCollector::UpdateState(GlParameterList &ResultCommands)

This operation is invoked when results of a request have come back from the dataserver indicating a change in state of the server side working collection.

The results are a set off commands. These are processed sequentially to change the state of the client-side collection to match that of the server side working collection.

This operation returns a GlStatus object.

```
DO WHILE (there are more results to process)
  Get the next command from the list of ResultCommands
  CASE (action portion of command)
    CASE_ADD:
      Create a DsClESDTReference from the data portion of the command
      Find the DsClESDTReferenceVector matching the type portion of the
          command
      Insert the DsClESDTReference into the DsClESDTReferenceVector
      Insert the DsClESDTReference into this DsClESDTReferenceCollector
    CASE_REPLACE:
      Find the DsClESDTReference matching the GlUR portion of the command
      Get the original DsClESDTReferenceVector from the DsClESDTReference
      Empty the DsClESDTReference of all information except its Collector
      Fill the DsClESDTReference from the data portion of the command
      IF (the original Type information differs from type portion of the
          command)
        Remove the DsClESDTReference from the original
                    DsClESDTReferenceVector
        Find the DsClESDTReferenceVector matching the type portion of the
            command
        Insert the DsClESDTReference into the DsClESDTReferenceVector
      END IF
    CASE_REMOVE:
      Remove the DsClESDTReference from its DsClESDTReferenceVector
      Delete the DsClESDTReferenceVector
  END CASE
END DO WHILE
```

~DsClESDTReferenceCollector
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClESDTReferenceCollector class has associations with the following classes:
None

4.3.13 DsClESDTReferenceVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This private class is a vector of ESDTReferences organized by type. This vector is intended to simplify management of ESDTs, which are presumed to be primarily accessed by type. This arrangement also permits storing TypeInfo only once per occurence of the given type.

Attributes:

myTypeInfo - This attribute holds all the information specific to a given type of ESDT. "Type" can mean product type, or newly-created type, as in the case of a subsetted ESDT.
Data Type: DsClTypeInfo*
Privilege: Private
Default Value:

Operations:

DsClESDTReferenceVector
Arguments:
Return Type: Void
Privilege: Public

DsClESDTReferenceVector
Arguments: DsClTypeInfo &
Return Type: Void
Privilege: Public

getTypeID - This operation returns the DsSdTypeID object for the given type. In the case of newly created types (e.g. a subsetted ESDT), this information may not mean anything (or even return anything).
Arguments:
Return Type: Void
Privilege: Public

getTypeInfo - This operation returns the DsClTypeInfo object for a set of ESDTReferences.
Arguments:

Return Type: Void
Privilege: Public

setTypeInfo - This operation sets the object's attribute to the given DsClTypeInfo object.
Arguments: DsClTypeInfo*
Return Type: Void
Privilege: Public

~DsClESDTReferenceVector
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClESDTReferenceVector class has associations with the following classes:
DsClESDTReferenceCollector (Aggregation)

4.3.14 DsClNotificationReceiver Class

Parent Class: Not Applicable
Public: No Distributed Object: Yes
Purpose and Description:
Handles asynchronous notifications from the data server to the currently logged-on client. Client will specify a callback point in his application code to receive these events (e.g. subscription triggering notifications) for each data server that s/he connects to.

Attributes:

myCallback - The callback that will be used when a notification is received.
Data Type: GlCallback
Privilege: Private
Default Value:

myReceiveFlag - Indicates whether notifications should be passed to callback; initially FALSE.
Data Type: RWBoolean
Privilege: Private
Default Value:

Operations:

DsClNotificationReceiver - Used to construct a notification receiver for a given server. Notifications will be sent to the given callback.
Arguments: server: GlUR, callback: GlCallback
Return Type: Void
Privilege: Public

DsClNotificationReceiver - Constructs a default notification receiver.
Arguments:
Return Type: Void
Privilege: Public

GetOneNotification - Used to wait for a notification to arrive. Starts a thread to wait, which can be stopped by StopReceiving().
Arguments:
Return Type: GlStatus
Privilege: Protected

SetCallback - Used to change the callback that is used when a notification arrives.
Arguments: callback: GlCallback
Return Type: void
Privilege: Public

StartReceiving - Used by the client to control when notifications may be received. Must be called after creation to enable reception of notifications.
Arguments:
Return Type: void
Privilege: Public

StopReceiving - Used to stop notifications from being sent to the callback. No notifications can be received in this state.
Arguments:
Return Type: void
Privilege: Public

~DsClNotificationReceiver - Used to destroy a notification receiver.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClNotificationReceiver class has associations with the following classes:
None

4.3.15 DsClQuery Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This public, local object simplifies the passing of query information from the client to the dataserver. The object is created in client space. The contents of the object will be used to create a Request object which will be passed to the dataserver. It is assumed that the "from" clause of an SQL query is inherent in specification of the dataserver to which the query is issued, that is, that the query is against "the inventory" of the dataserver. Any conversion to actual table names which may be necessary is done transparently to the client software.

Attributes:

AttributesToReturn - This attribute represents the projection of the query (i.e. the "select" clause from an SQL statement). It is implemented as a GlParameterList because it has additional usage in the "inspecting" of the results of the query.
Data Type: GlParameterList
Privilege: Private
Default Value: ("all", char, "*")

Constraints - This attribute represents the constraints of a query (i.e. the "where" clause of an SQL statement).
Data Type: GlParameterList
Privilege: Private
Default Value:

MaxHits - This attribute indicates the maximum number of granules to return (regardless of the number of actual hits, i.e. the granules which satisfy the search criteria). Perhaps in the future, this can be used to indicate the amount to check "estimated hits" against.
Data Type: int
Privilege: Private
Default Value:

QueryType - This attribute controls the scope of the query. DsTQueryType is an enum, where the possible query types are Inventory and WorkingCollection. Issuing a query against the inventory results in a search of the Inventory object. Issuing a query against the

WorkingCollection results in a refinement of the existing collection. Issuing a query against an empty WorkingCollection will return null.
Data Type: DsTQueryType
Privilege: Private
Default Value: Inventory

myLocalCallback - This attribute identifies the local callback routine to be invoked when any status is returned relative to this query. This attribute is passed to the DsClRequest object which is created from the DsClQuery object. The DsClRequest object will use this information to relate status from the DsClSubmittedRequest to the DsClRequest object. As far as the DsClQuery object is concerned, this is just a pass-through.
Data Type: GlCallback
Privilege: Private
Default Value:

myStatus - This attribute allows the object to maintain information on current status.
Data Type: GlStatus
Privilege: Private
Default Value:

Operations:

ConvertFromSQL - This operation is to support incoming SQL queries, in the case that some client software somewhere generates only SQL.
Arguments: RWCString &
Return Type: GlStatus &
Privilege: Public

ConvertToCommand - This operation converts the information in the Query object to a command format that is interpretable by the dataserver. The command is then packaged in a Request object prior to returning.
Arguments: DsClRequest &
Return Type: DsClRequest &
Privilege: Protected

ConvertToSQL - This operation provides the DsClQuery object in SQL format to support client software that can only deal with SQL queries.
Arguments: RWCString &
Return Type: RWCString &
Privilege: Public

DsClQuery
Arguments:

Return Type: Void
Privilege: Public

DsClQuery - This version of the constructor takes the Constraints in the form of a ParameterList and uses the default for the AttributesToReturn.
Arguments: GlParameterList &
Return Type: Void
Privilege: Public

GetAttributesToReturn - This operation returns the currently defined set of display attributes for a search.
Arguments: GlParameterList &
Return Type: GlStatus &
Privilege: Public

GetCallback - This operation allows the client software to get the current setting of the query callback.
Arguments: GlCallback &
Return Type: GlStatus &
Privilege: Public

GetConstraints - This operation returns the currently set list of user-defined query constraints.
Arguments: GlParameterList &
Return Type: GlStatus &
Privilege: Public

GetMaxHits - This operation returns the current setting of MaxHits. For Release A, MaxHits indicates the maximum number of rows to return (regardless of the actual number found by the query). This is in keeping with the definition of MaxHits in V0. In the future, it may be used to indicate the cutoff limit for running a query, i.e. if the estimated number of hits is greater than MaxHits, then don't run the query.
Arguments: int &
Return Type: GlStatus &
Privilege: Public

GetQueryType - This operation returns the current setting for the QueryType.
Arguments:
Return Type: DsTQueryType
Privilege: Public

SetAttributesToReturn - This operations allows the client software to define a subset of available attributes as the display attributes for the result of a search.
Arguments: GlParameterList &

Return Type: GlStatus &
Privilege: Public

SetCallback - This operation allows the client software to specify a local callback for queries so that the DsClESDTReferenceCollector object (which is distributed) can return interim status information to the client software.
Arguments: GlCallback &
Return Type: GlStatus &
Privilege: Public

SetConstraints - This operation allows the client software to pass a list of user- specified query constraints.
Arguments: GlParameterList &
Return Type: GlStatus &
Privilege: Public

SetMaxHits - This operations allows the client software to specify the number of rows to return in response to a query (this per the current definition of MaxHits in V0). Perhaps in the future it can be used to specify the number of hits to compare to a query estimator; in that case the query will not execute if the anticipated number of result objects is greater than the value in MaxHits.
Arguments: int
Return Type: GlStatus &
Privilege: Public

SetQueryType - This operation sets the scope of the query, i.e., run against the Inventory or against the WorkingCollection.
Arguments: DsTQueryType
Return Type: GlStatus &
Privilege: Public

~DsClQuery
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClQuery class has associations with the following classes:
None

4.3.16 DsClRequest Class

Parent Class: DsSrRequestBase
Public: No Distributed Object: No
Purpose and Description:
A specialization of DsRequest for client interfaces. Allows the client to compose a request and submit it to the data server. Once submitted, the status may be polled, or a callback can be provided that is triggered on every status change.

Attributes:

myCallback - Callback that will be made anytime the status changes.
Data Type: GlCallback
Privilege: Private
Default Value:

myCollector - Reference to the collector to which this request belongs.
Data Type: DsClESDTReferenceCollector *
Privilege: Private
Default Value:

myQuery - This is the query object associated with this request. The query object is needed during a search request because the collector needs to look up the callback for the query after it is called-back by the request when the search status changes.
Data Type: DsClQuery *
Privilege: Private
Default Value: NULL

mySubmittedFlag - Indicates whether this request has been submitted.
Data Type: RWBoolean
Privilege: Private
Default Value: FALSE

Operations:

Cancel - Used to cancel a request. If the request is not executing (i.e. it's queued), then it is removed from the queue. If it is executing, it will be terminated after the current command completes.
Arguments:
Return Type: GlStatus
Privilege: Public

DsClRequest - Used to construct an "empty" request, which should be filled-in by calling SetPriority (and other functions) and adding commands.
Arguments:
Return Type: Void
Privilege: Public

DsClRequest - Constructs a request with the given command and priority.
Arguments: cmd: DsClCommand *, pty: DsESrRequestPriority
Return Type: Void
Privilege: Public

GetQuery - Returns the current query object associated with this request.
Arguments:
Return Type: DsClQuery *
Privilege: Public

GetResults - Returns the current results list from the associated DsClSubmittedRequest.
Arguments:
Return Type: const GlParameterList &
Privilege: Public

SetQuery - Used to set the query object associated with this request. The query object is needed during a search request because the collector needs to look up the callback for the query after it is called-back by the request when the search status changes.
Arguments: DsClQuery *
Return Type: void
Privilege: Public

SetStatusCallback - Used to provide a client entry point to be called on every change of the status of the request.
Arguments: GlCallback *
Return Type: void
Privilege: Public

Submit - Used to submit a request to be executed by a single, specific ESDT. The request is in turn submitted to the "implied" DsClESDTReferenceCollector, i.e. the one the DsClESDTReference holds a pointer to.
Arguments: DsClESDTReference &
Return Type: GlStatus
Privilege: Public
PDL:IF this request has already been submitted
RETURN code indicating request-already-submitted
ELSE
CALL DsClESDTReference::GetCollector to obtain the collector that contains it
CALL GlURVector to create a new domain vector

CALL DsClESDTReference::GetUR to obtain the ESDT's UR
CALL GlURVector::insert to add the ESDT's UR to the domain vector
CALL DsClRequest::Submit with the collector and domain vector
END IF

Submit - Used to submit a request that is to be executed over an entire collector, e.g. a search that adds/removes ESDTs or operation(s) to be performed by all of the ESDTs. A request can be submitted to a specific subset of ESDTs in the collection by passing a GlURVector with the URs of the desired ESDTs.
Arguments: DsClESDTReferenceCollector &, GlURVector * = NULL
Return Type: GlStatus
Privilege: Public
PDL:IF this request has already been submitted
RETURN code indicating request-already-submitted
ELSE
IF the domain vector pointer != NULL
CALL DsClRequest::SetDomain to set the domain for this request
END IF
CALL DsClESDTReferenceCollector::SubmitToServer to send this request to the server
END IF

Textify - Used to convert a request into a human-readable format.
Arguments:
Return Type: RWCString
Privilege: Public

~DsClRequest - Used to destroy a request.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClRequest class has associations with the following classes:
DsClAction (Aggregation)

4.3.17 DsClRequestVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No

Purpose and Description:
An instantiation of RWVector over DsClRequest.

Attributes:

None

Operations:

None

Associations:

The DsClRequestVector class has associations with the following classes:
DsClESDTReferenceCollector (Aggregation)

4.3.18 DsClSubmittedRequest Class

Parent Class: Not Applicable
Public: No Distributed Object: Yes
Persistent Class: True
Purpose and Description:
This class is used when a client request is submitted to the data server. It represents the "distributedness" of a request, and manages communication between the real request on the server and the client's request. It provides status to the DsClRequest when the client inquires for it, or by using the callback. It also gives the client access to the results of the request execution through DsClRequest.

Attributes:

myCallback - On the client side, this is the callback that is used every time the status changes for this request.
Data Type: GlCallback
Privilege: Private
Default Value:

myConnection - The ID of the connection with which this request is associated.
Data Type: DsESrConnectionID
Privilege: Private
Default Value:

myFinishTime - The date and time on which this request finished.
Data Type: RWTime
Privilege: Private
Default Value:

myID - The system-wide ID for this submitted request.
Data Type: DsClSubmittedRequestID
Privilege: Private
Default Value:

myInfo - This is the core information for this request.
Data Type: DsSrRequestInfo*
Privilege: Private
Default Value:

myResults - This is a list of results returned from the execution of a request. The parameters in the list are really other GlParameterList's, one for each command in the request. Each of these is composed of GlParameterList's, one for each ESDT in the collection. It is filled in by the DsSdWorkingCollection.
Data Type: GlParameterList
Privilege: Private
Default Value:

myStatus - The current status of the request, updated as the request moves through the data server and executes. A mirror of the status field in the real request.
Data Type: GlStatus
Privilege: Private
Default Value:

mySubmitTime - The date and time on which this request was submitted.
Data Type: RWTime
Privilege: Private
Default Value:

myUser - The user ID of the user who submitted the request.
Data Type: MSS_UserID
Privilege: Private
Default Value:

Operations:

DsClSubmittedRequest - Constructs a submitted request with the given connection ID and user ID. The DsClRequest is used to fill-in myInfo. mySubmitTime and myUR are automatically filled-in from information available from the system.
Arguments: DsESrConnectionID, MSS_UserID, DsClRequest&
Return Type: Void
Privilege: Public

GetConnectionID - Used to retrieve the ID of the connection with which the request was submitted.
Arguments:
Return Type: DsESrConnectionID
Privilege: Public

GetFinishTime - Used to obtain the date and time on which this request finished execution.
Arguments:
Return Type: RWTime
Privilege: Public

GetID - Returns the ID of this submitted request.
Arguments:
Return Type: DsClSubmittedRequestID
Privilege: Public

GetInfo - Used to retrieve a pointer to the core information for this request. DsSrConnection (and its sub-classes) use this to construct each DsSrRequest.
Arguments:
Return Type: DsSrRequestInfo*
Privilege: Protected

GetResults - Returns a reference to the current results list. Results are updated on the server side.
Arguments:
Return Type: GlParameterList &
Privilege: Public

GetStatus - Returns the current status of the request.
Arguments:
Return Type: const GlStatus &
Privilege: Public

GetSubmitTime - Used to obtain the date and time on which this request was submitted.
Arguments:

Return Type: RWTime
Privilege: Public

GetUser - Used to retrieve the ID of the user that submitted the request.
Arguments:
Return Type: MSS_UserID
Privilege: Public

SetCallback - Sets the value of the callback that is called every time the status of the request changes.
Arguments: GlCallback
Return Type: void
Privilege: Public

SetPriority - Used by client software to set the priority of a request that has been submitted. On the server side, access the core request information directly to set the priority.
Arguments: DsESrRequestPriority
Return Type: void
Privilege: Public

SetStatus - Used to set the status of the request. Used exclusively on the server side.
Arguments: GlStatus *
Return Type: void
Privilege: Public

WaitStatus - Used on the client side to initiate a thread that waits for a change in status in the request. Whenever a change occurs (on the server side), this call will return and the client may check its value.
Arguments:
Return Type: void
Privilege: Public

~DsClSubmittedRequest - Used to destroy a submitted request.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClSubmittedRequest class has associations with the following classes:
Class: DsSrRequestBase

4.3.19 DsClSubscription Class

Parent Class: Not Applicable
Public: Yes Distributed Object: No
Purpose and Description:
This class is the client side subscription which can either be created from advertisements or from exisiting subscriptions from the server side (through a stream.)

Attributes:

myAction - The action to be performed when the subscription fires.
Data Type: DsClAction
Privilege: Protected
Default Value:

myCollector - A pointer to the collector that this reference is a member of. If this pointer is null, then this reference is a member of one of the collectors in the static collector vector.
Data Type: DsClSubscriptionCollector&
Privilege: Protected
Default Value:

myDescription - String which contains service of the subscription.
Data Type: RWCString
Privilege: Protected
Default Value:

myDurationType - Time duration of subscriptions (i.e., can be done one time or forever (outstanding).
Data Type: enum DsEClSubscriptionType
Privilege: Protected
Default Value: {ONCE, OUTSTANDING}

myExpirationDate - Identifies when this subscription will expire and be removed from the system. The value may be "never" (i.e. the subscription is permanent)
Data Type: RWDate
Privilege: Protected
Default Value:

mySubmittedFlag - Flag which shows whether the subscription has been submitted or not.
Data Type: RWBoolean
Privilege: Protected
Default Value: RWTrue

myUserInfo - Client information, provided by client software.
Data Type: DsClClient&
Privilege: Protected
Default Value:

ourCollectorvector - Static vector of pointers to DsClSubscriptionCollector objects, one per dataserver.
Data Type: DsClSubscriptionCollectorVector &
Privilege: Protected
Default Value:
This is a Class Attribute.

Operations:

DsClSubscription
Arguments:
Return Type: Void
Privilege: Public

DsClSubscription - Constructor for client software (therefore, public) which gets attribute information from advertisements, such as service provider. If no collector has been provided, it goes and finds one, based on the static nature of the collector.
Arguments: userinfo, Advertisement&, DsClSubscriptionCollector&
Return Type: Void
Privilege: Public

DsClSubscription - Constructor for already existing collector which gets already existing subscriptions from the sever side through a stream.
Arguments: submittedflag, DsClSubscriptionCollector&, Stream
Return Type: Void
Privilege: Public

GetAction - Means of accessing myAction attribute (object), which will be communicated to the server what this subscription should do when it fires.
Arguments: DsClAction &
Return Type: void
Privilege: Public

GetDescriptoin - Returns description, containing the service, as a roquewave string.
Arguments:
Return Type: RWCString
Privilege: Public

GetDurationtype - Accesses myDurationType attribute as to whether subscriptions are done one time or forever (outstanding).
Arguments:
Return Type: DsEClSubscriptionType
Privilege: Public

GetExpirationdate - Public access to myExpirationDate attribute, which provides the expiration date of the subscription.
Arguments:
Return Type: RWDate
Privilege: Public

GetSubmittedflag - Public access to flag as to whether a subscription has been submitted.
Arguments:
Return Type: RWBoolean
Privilege: Private

GetUserinfo - Public access to user information which can be put into the DsSrClient object.
Arguments: GLClient&
Return Type: Void
Privilege: Public

SetAction - Sets the myAction attribute for this particular subscription as determined by the client software.
Arguments: DsClAction&
Return Type: void
Privilege: Public

SetDescription - Allows the client software to fill in the Description attribute with service information.
Arguments: RWCString
Return Type: void
Privilege: Public

SetDurationType - Sets the attribute which determines the existence type of the subscription.
Arguments: DsEClSubscriptionType
Return Type: void
Privilege: Public

SetExpirationDate - Sets the expiration date of the subscription itself.
Arguments: RWDate
Return Type: void
Privilege: Public

SetSubmittedFlag - Sets the flag which indicates whether or not the DsClSubscription has actually been submitted to the dataserver (i.e. the client software is finished with filling in the information, and has invoked the Submit method).
Arguments: RWBoolean
Return Type: void
Privilege: Public

Submit - Submits subscription to the subscription collector.
Arguments:
Return Type: GlStatus&
Privilege: Public

Withdraw - Deletes a subscription from the subscription collector.
Arguments:
Return Type: GlStatus&
Privilege: Public

~DsClSubscription
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClSubscription class has associations with the following classes:
None

4.3.20 DsClSubscriptionCollector Class

Parent Class: DsClCollector
Public: No Distributed Object: Yes
Purpose and Description:
This public, distributed class is a specialization of the Collector class which handles DsClSubscriptions. This class provides, in addition to the normal vector operations, the ability to create a list of all subscriptions for a given user or advertisement, and a means of submitting and cancelling subscriptions. There are no attributes for this object.

Attributes:

myStatus - This attribute allows the object to maintain information on current status.
Data Type: GlStatus
Privilege: Private
Default Value:

Operations:

BuildList - This operation creates a list of all subscriptions for a given event.
Arguments: Advertisement&
Return Type: const GlStatus &
Privilege: Public

BuildList
Arguments: MSS_UserProfile &
Return Type: const GlStatus &
Privilege: Public

CancelSubscription - This operation creates a request to cancel the specified subscription.
Arguments: DsClSubscription*
Return Type: const GlStatus &
Privilege: Private

CreateSubscription
Arguments: RWBoolean SubmittedFlag, istream &Stream, DsClSubscriptionCollector *me
Return Type: DsClSubscription*
Privilege: Private

DsClSubscriptionCollector
Arguments: GlUR &dataserver, MSS_UserProfile &
Return Type: Void
Privilege: Public

~DsClSubscriptionCollector
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsClSubscriptionCollector class has associations with the following classes:
None

4.3.21 DsClTypeInfo Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This private, local object provides the place for information related to a type of ESDT to be stored. This information all comes from the Descriptor. This information will be stored only once per Descriptor (i.e. product type) and referenced by all the ESDTReferences of that type. This object contains a DsSdTypeID, which provides the name and version for this type.

Attributes:

myMetadata - This attribute holds the metadata attributes which apply to all objects of the given type. This metadata is both the core and the product specific metadata for this granule.
Data Type: GlParameterList
Privilege: Private
Default Value:

myQueryableParameters - This attribute contains the list of attribute names for ESDT's of this type. This list allows the client software to determine which attributes to use in certain services, such as Inspect.
Data Type: GlParameterList
Privilege: Private
Default Value:

myScienceParameters - This attribute holds the Science Parameters from the Descriptor for the ESDTReferences of this type. These are available for the client software to query and use in constructing parameter lists for advertised services.
Data Type: GlParameterList
Privilege: Private
Default Value:

myStatus - This attribute allows the object to maintain information on current status.
Data Type: GlStatus

Privilege: Private
Default Value:

Operations:

GetMetadata - This operation returns the GlParameterList which represents all the metadata attributes for this granule, both Core and Product Specific.
Arguments: GlParameterList &
Return Type: GlStatus &
Privilege: Public

GetName - This operation returns the name from the DsGlTypeID object embedded in this DsClTypeInfo object.
Arguments: RWCString &
Return Type: GlStatus &
Privilege: Public

GetQueryableParameters - This operation allows the client software to retrieve the list of queryable parameters which apply to a specific ESDTReference. The information is applicable to all ESDT's of the same type.
Arguments: GlParameterList &
Return Type: GlStatus &
Privilege: Public

GetScienceParameters - This operation returns the ParameterList which represents the Science Parameters from the associated Descriptor.
Arguments: GlParameterList &
Return Type: GlStatus &
Privilege: Public

GetVersion - This object returns the version from the DsSdTypeID object embedded in this DsClTypeInfo object.
Arguments: RWCString &
Return Type: GlStatus &
Privilege: Public

Associations:

The DsClTypeInfo class has associations with the following classes:
DsClESDTReferenceVector (Aggregation)

4.3.22 DsCnConfiguration Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class: True
Purpose and Description:
Manages configuration parameters for various data server components in a token-value style file. Often specialized where needed to provide higher level access to configuration parameters.

Attributes:

myFile - The file with which this configuration object is associated.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

Add - Add a token value to a configuration file under any section.
Arguments: const RWCString&, const RWCString&
Return Type: RWBoolean
Privilege: Public

Add - Add a token value to a configuration file under a specific section.
Arguments: const RWCString&, const RWCString&, const RWCString&
Return Type: RWBoolean
Privilege: Public

DsSdConfiguration
Arguments: const RWCString&

Get - Used to acquire a token value out of a file. Meant to be used as a general means to get tokens out of files, but in this case is used for configuration-like files.
Arguments: const RWCString&
Return Type: const RWString
Privilege: Public

Get - Used to acquire a token value from any configuration-like file. This particular function takes care of the case when a specific section is specified.
Arguments: const RWCString&, const RWCString&

Return Type: const RWString
Privilege: Public

GetSectionToken - This is a general section parser from a file. It can parse multiple sections from a file line.
Arguments: const RWCString&, RWCString&
Return Type: RWBoolean
Privilege: Protected

ReDoFile - Rewrites a file at the system level after lines have been modified, added, or deleted.
Arguments:
Return Type: const RWBoolean
Privilege: Protected

Remove - Removes a line matching an input token key under any section.
Arguments: const RWCString&
Return Type: RWBoolean
Privilege: Public

Remove - Remove a configuration file line matching an input token key under a specific section.
Arguments: const RWCString&, const RWCString&
Return Type: RWBoolean
Privilege: Public

Update - Update a line in a configuration file given a specific key with which to update. This occurs for any section.
Arguments: const RWCString&, const RWCString&
Return Type: RWBoolean
Privilege: Public

Update - Update a line in a configuration given an input key under a specific section.
Arguments: const RWCString&, const RWCString&, const RWCString&
Return Type: RWBoolean
Privilege: Public

~DsSdConfiguration
Arguments:

Associations:

The DsCnConfiguration class has associations with the following classes:
None

4.3.23 DsCnDSSConfiguration Class

Parent Class: DsCnConfiguration
Public: No Distributed Object: No
Purpose and Description:
This is a datserver configuration startup class that starts processes as defined in a configuration file.

Attributes:

All Attributes inherited from parent class

Operations:

GetProcessesToStart - Uses the base configuration class to read processes from a file in order to initiate processes in that file.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCnDSSConfiguration class has associations with the following classes:
Class: DsCnDSSStartup enquires

4.3.24 DsCnDSSStartup Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This starts up all processes for a particular dataserver by enquiring a particular configuration file and initializing/restarting ESDTs.

Attributes:

None

Operations:

InitializeESDTs - Creates descriptors for each type in configuration.
Arguments:
Return Type: RWBoolean
Privilege: Public

RestartESDTs - If something should happen to the advertisement server, then redo the initialize ESDTs and "re-advertise" them.
Arguments:
Return Type: RWBoolean
Privilege: Public

StartProcesses - This is the means to interface with an MSS object which takes care of initiating startup and shutdown procedures from a system-level perspective.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCnDSSStartup class has associations with the following classes:
Class: DsCnDSSConfiguration enquires
Class: DsGeESDTConfiguration enquires
Class: DsDeESDTDescriptor initializes

4.3.25 DsCoCombination Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
The class represents 'combined' products, generated using data from more than one satellite or ground based instrument. An example would be the TRMM 2B-31 Level 2B product generated using data from the PR or/and VIRS TRMM instruments.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsCoCombination class has associations with the following classes:
None

4.3.26 DsDbAccess Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class: True
Purpose and Description:
This object provides a layer of database-technology-independence to the dataserver objects. Each object type in the dataserver is defined in this object (i.e. a given instantiation of this object will collect the appropriate information about the provided object type from the database and use that information to extract the appropriate data from the database to fill in the attributes of the given object).

Attributes:

myAssociations - This attribute contains a list of table names with which the given CollectableObject is associated. This attribute is assumed to be a one-to-one mapping with the values in myForeignKeys, and it is further assumed that the order in which the associated tables appear corresponds with the order of column names contained in attribute myForeignKeys.
Data Type: RWTPtrOrderedVector<RWCString>
Privilege: Private
Default Value:

myDBConnections - This static attribute holds a set of database connections so that each instantiation of the object can have immediate access to the database (i.e., without having to login and open the database). This pool of database handles is established at dataserver startup, and should have a configurable limit to be stored (that is, the actual number of connections to create should be a configuration variable). This pool should be removed

(each connection closed) at destructor time. In case of a crash, Sybase will probably clean up broken connections anyway, so this case can be ignored. If Sybase does NOT clean up broken connections, then whatever process is the norm for doing so (manual or otherwise) can be used.
Data Type: RWTPtrOrderedVector<DsDbInterface>
Privilege: Private
Default Value:
This is a Class Attribute.
This is a Derived Attribute.

myDatabase - This is a derived attribute in that it is not stored anywhere. It is static because all objects in the Dataserver will be stored in the same database. This attribute is used to open the database after a connection (login) to the server has been established.
Data Type: RWCString & DBName
Privilege: Private
Default Value:
This is a Class Attribute.
This is a Derived Attribute.

myForeighKeys - This attribute contains a list of column names which are the primary keys for the tables with which the given CollectableObject is associated. This attribute is assumed to be a one-to-one mapping with the values in myAssociations, and it is further assumed that the order in which the column names appear corresponds with the order of associated tables contained in attribute myAssociations.
Data Type: RWTPtrOrderedVector<RWCString>
Privilege: Private
Default Value:

myIndexableColumns - This attribute contains a list of column names indicating which of the attributes of the given CollectableObject type are stored separately from the rest of the object. This allows indexes to be built on these fields. It also allows human perusal of the values. It also allows joins on these fields. The fields in this list do not necessarily have indexes built on them, as they may appear hear for one of the latter two reasons.
Data Type: RWTPtrOrderedVector<RWCString>
Privilege: Private
Default Value:

myLocation - This attribute identifies either the table name (for objects which are stored in a DBMS) or the file (?path) name (for objects which are stored as files).
Data Type: RWCString
Privilege: Private
Default Value:

myObjectIdentifier - This attribute is used for those accesses which require the storing or retrieval of a specific object (i.e. a specific instantation of the given class). The value in

this object is used as a qualifier in selecting from the underlying database.
Data Type: int
Privilege: Private
Default Value:

myObjectType - This attribute indicates which of the dataserver objects is being handled at this time. The object is known only by it's ancestor type CollectableObject at the time that this object (DsDbAccess) is created. Each CollectableObject must be capable of identifying its type. The value of this attribute allows the DsDbAccess object to know the values for the rest of the attributes. Based on the ObjectType, DsDbAccess can read the corresponding control information (rest of attributes) from the underlying database.
Data Type: DsEDbObjectType
Privilege: Private
Default Value:

myPersistenceType - This attribute identifies whether the given CollectableObject is to be stored in a DBMS or in a file.
Data Type: DsEDbPersistenceType
Privilege: Private
Default Value:

myPrimaryKey - This attribute identifies which of the given CollectableObject's attributes is to be used as a Primary Key for selecting information about this type of object. This attribute will store the DBMS column name of the PK column in table "myLocation".
Data Type: RWCString
Privilege: Private
Default Value: myObjectIdentifier

myStuff - This attribute represents the given CollectableObject type in binary. This attribute is not human readable, can not be indexed, and can not be used in joins.
Data Type: binary
Privilege: Private
Default Value:

Operations:

CloseDatabase - This protected operation allows this object to close all database connections on system shutdown. This is a class operation because the pool of database handles is static.
Arguments:
Return Type: const GlStatus &
Privilege: Protected
This is a Class Operation.

DsDbAccess - The default constructor allows users of this object to create an empty one in the case where specific values are to be found. An empty DsDbAccess object is created, then the actual values in the CollectableObject which identify the data to be used can be identified at a later time.
Arguments:
Return Type: Void
Privilege: Public

DsDbAccess - This version of the constructor takes the partially filled in CollectableObject (which contains, at a minimum, the object type), and fills the object attributes prior to returning.
Arguments: CollectableObject &
Return Type: Void
Privilege: Public

Fill - This operation supplies this object with the partially filled-in CollectableObject such that this object (DsDbAccess) can find the data for the attributes of the CollectableObject. For example, a given CollectableObject may contain an attribute called "name" which is used to identify which instance is desired. The DsDbAccess object will find out what the type of the Collectable Object is, get the definition information for that type of object, identify that "name" is the primary key, and then construct a query using the supplied value for "name" to get the rest of the data which corresponds to that name. This operation returns the entire object in a binary stream. The calling object is responsible for interpreting the stream according to its internal structure.
Arguments: const CollectableObject &, ostream &
Return Type: const GlStatus &
Privilege: Public
PDL:DsDbAccess::Fill(CollectableObject&, ostream &)

This operation is invoked when a previously-stored dataserver object is created. The calling operation (which may be the constructor of the object) provides the reference to the object. The calling operation has filled in those attributes (from among the ones defined as "indexable") which will be used to identify this object.

This operation uses the configuration information for the given object type and the values supplied for this instance of the object type to construct a database query. The configuration information includes an indicator of whether or not an object can receive more than one result from the database. The results of the query are sent to the stream provided as an input parameter. The calling operation must have an operation to reconstruct as necessary from a stream.

This operation puts the results of the query via the stream parameter and returns a GlStatus object. If an object is defined as unit-level (that is,

"filling" the object involves finding a single entry in the database) and the results of the query are not unit-level, the status will indicate an error.

```
SEQUENCE:
  Get the object's type
  Get the configuration information for that object type
  Construct the projection of the query from the configuration information,
      i.e., ("Select ", Config->myStuff)
  Construct the location of the query from the configuration information,
      i.e., ("from ", Config->Location)
  Construct the restriction of the query from the supplied attributes,
      i.e., ("where ")
  FOR (each of the type's indexable columns)
    IF (the supplied attribute is not null)
      IF (this is not the first constraint)
        Append a conjuctive to the query,
            i.e., (" AND ")
      END IF
      Append a constraint to the query
      EXAMPLE:
        (Config->myIndexables->thisColumn,
         Config->myIndexables->operator,
                    //(" = ", or " > ", etc.)
         CollectableObject->(thisColumn))
       becomes
         (TypeCode, "=", Object->TypeID->myCode)
       becomes
         "TypeCode = 5"
      END EXAMPLE
    END IF
  END FOR
  Create a DsDbInterface object
  Submit the query to the DsDbInterface object
  IF (the configuration information indicates "set" level return values)
    Get the results from the DsDbInterface routine which returns a vector of
        results
    Bind the return vector to the stream parameter supplied to this operation
  ELSE
    Get the results from the DsDbInterface routine which returns a single
       result
    Bind the return result to the stream parameter supplied to this operation
  END IF
  Set the GlStatus object to reflect the current status
  Return the GlStatus object
END SEQUENCE
```

GetDBHandle - This internal operation is used by the object on construction, to get the next available DB handle from the static pool of handles. This handle is then used for all the database operations performed by this instantiation.
Arguments:
Return Type: DsDbConnection
Privilege: Protected

NextTypeCode - This operation gets a unique code for a new (previously not existing) TypeID.
Arguments:
Return Type: int
Privilege: Public

OpenDatabase - This protected operation allows this object to establish a pool of connections to the database upon system startup. This is a class operation because the pool of database handles is static.
Arguments:
Return Type: const GlStatus &
Privilege: Protected
This is a Class Operation.

ReturnDBHandle - This internal operation is used by the object on destruction, to return the handle it used to the pool of available handles, so that another instantiation can use it.
Arguments: DsDbInterface
Return Type: const GlStatus &
Privilege: Protected

Store. - This operation takes the data in the provided CollectableObject and inserts it to the database according to the definition information which is recorded for the CollectableObject's type. This operation returns the ObjectID which is assigned by the DBMS.
Arguments: const CollectableObject &
Return Type: int
Privilege: Public
PDL:DsDbAccess::Store(CollectableObject&)

This operation is invoked when a dataserver object which is identified as being persistent is created in full (i.e. has all attributes set, as opposed to just having the object constructed). The calling operation (which may be the constructor of the object) provides the reference to the object.

This operation uses the configuration information for the given object type and the values supplied for this instance of the object type to construct a database insert statement.

This operation returns a GlStatus object.

```
SEQUENCE:
  Get the object's type
  Get the configuration information for that object type
  Construct the projection of the statement from the configuration information,
      i.e., ("Insert ")
  Construct the location of the statement from the configuration information,
      i.e., (Config->Location)
  Construct the restriction of the statement from the supplied attributes,
      i.e., ("values ")
  FOR (each of the type's indexable columns)
    Append a value to the query,    i.e., ("XXX, ")
  END FOR
  Create a DsDbInterface object
  Submit the statement to the DsDbInterface object
  Set the GlStatus object to reflect the current status
  Return the GlStatus object
END SEQUENCE
```

UnStore - This operation removes the information of the current object from the persistence store. It does *not* remove the object from existence, that is, it simply makes the object volatile, that is, not persistent. The local object which has been "UnStore"d should not use any persistence-related attributes (ObjectID) or make any assumptions that previously associated objects will continue to "know" about this object. This bi-level removal of an object from the persistent store is to provide greater control in the object itself, that is, an object may have reason to be not-persistent at some point.
Arguments: CollectableObject &
Return Type: const GlStatus &
Privilege: Public

Update - This operation allows the object to change values of attributes without changing the identity of the object itself. Attempting to change the attribute(s) which constitute the primary key for the object is an error.
Arguments: const CollectableObject &, istream &
Return Type: const GlStatus &
Privilege: Public

~DsDbAccess - The destructor closes the database connection (returns the handle) and anything else that is necessary.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDbAccess class has associations with the following classes:
Class: DsDbInterface

4.3.27 DsDbAttributeToTableVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
This class contains the mappings of attributes to tables and columns within the database. This class contains default locations for each attribute but also allows the table and column names for an attribute to be overloaded on a product basis.

Attributes:

myAttributeToTableVector - This vector holds the mappings of attributes to table and column names.
Data Type: DsTMdAttributeTableXrefList
Privilege: Private
Default Value: null

Operations:

GetTableColumnName - This method takes a DsTMdAttributeXref structure with the product and attribute name filled in and updates the stucture adding the table and column names.
Arguments: DsTMdAttributeTableXref& attribute, DsMdDbConnection& connection
Return Type: GlStatus
Privilege: Public

Initialize - This method queries the DsMdAttributeTableXref table and builds the myAttributeToTableVector. This vector will hold all the mappings for the Catalog.
Arguments: DsMdDbConnection& connection
Return Type: GlStatus
Privilege: Public

PutTableColumnNames - This method adds a row to the DsMdAttributeTableXref table using the information passed in through the DsTMdAttributeXref structure.
Arguments: DsTMdAttributeTableXref& newAttribute, DsMdDbConnection& connection
Return Type: GlStatus
Privilege: Public

UpdateTableColumnName - This method uses the product and attribute values in the DsTMdAttributeXref structure to identify a row in the DsMdAttributeTableXref table and update it to the table and column names provided in the structure.
Arguments: DsTMdAttributeTableXref& attribute, DsMdDbConnection& connection
Return Type: GlStatus
Privilege: Public

Associations:

The DsDbAttributeToTableVector class has associations with the following classes:
Class: DsDbInterface

4.3.28 DsDbEngine Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class represents the Sybase SQL server COTS product.

Attributes:

None

Operations:

None

Associations:

The DsDbEngine class has associations with the following classes:
Class: DsDbInterface

4.3.29 DsDbGranuleToDbVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
This class holds the mappings of granules to databases. The class allows for granules to be partitioned based upon product type and temporal range.

Attributes:

myGranuleToDBVector - This attribute holds the mappings of granules to database names.
Data Type: DsMdProductDbXrefList
Privilege: Private
Default Value: NULL

Operations:

GetProductDb
Arguments: DsTMdProductDbXref& product, DsTMdProductDbXrefList& result, DsMdDbConnection& c onnection

Initialize - This method queries the DsMdProductDbXref table and populates the myGranuleToDbList with all the mapping for the catalog.
Arguments: DsMdConnection& connection
Return Type: GlStatus
Privilege: Public

PutProductDb - This method adds a new product to database row to the DsMdProductDbXref table.
Arguments: DsTMdProductDbXrefList& productDbList, DsMdDbConnection& connection
Return Type: GlStatus
Privilege: Public

UpdateProductDb - This method updates the rows of the DsMdProductDbXref table associated with the product names supplied in the DsTMdProductDbXref structures in the supplied list. The database name columns are changed to the database names in the structures. If a DsTMdProductXref item doesn't exist in the table, it is added. This method

operates on a list to assure that no overlap exists in the date ranges for a given product.
Arguments: DsTMdProductDbXrefList& productDbList, DsMdDbConnection& connection
Return Type: GlStatus
Privilege: Public

Associations:

The DsDbGranuleToDbVector class has associations with the following classes:
Class: DsDbInterface

4.3.30 DsDbInterface Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This object provides a lower level database wrapper for system applications. It makes all the product-dependent database client library calls transparent to the high level user. It is mainly responsible for establishing, maintaining database connection, executing utility query (e.g. open database, begin/end transaction) or user query (which returns data). The result of user query can be retrieved one at a time through a generic interface to bind result attribute data to user variables.

Attributes:

myCommand - Handle for last command executed. This is a COTS product specific item (e.g. Sybase). It is hidden from caller for product transparency. The SQL statement specified by caller during ExecuteQuery() call is embedded inside the command structure.
Data Type: CS_COMMAND*
Privilege: Private
Default Value: null

myConnection - Handle to the current database connection. This is a COTS specific (in this case - Sybase) data structure. It is hidden from the caller for product transparency.
Data Type: CS_CONNECTION*
Privilege: Private
Default Value: null

myContext - Database client-library programming context.
Data Type: CS_CONTEXT*

Privilege: Private
Default Value: null

myExecStatus - Current database command execution status.
Data Type: DsTDbExecStatus
Privilege: Private
Default Value: EXEC_NONE

myPassWord - User password for database connection. It is expected to be set during the Connect() operation.
Data Type: char*
Privilege: Private
Default Value: null

myServerName - Server name for database connection (e.g. SYBASE for default Sybase server). It is expected to be set in the Connect() operation.
Data Type: char*
Privilege: Private
Default Value: null

myState - Current database connection state.
Data Type: DsTDbConnectionState
Privilege: Private
Default Value: NOT_CONNECTED

myUserName - User id for database connection. It is expected to be set during the Connect() operation.
Data Type: char*
Privilege: Private
Default Value: null

Operations:

Connect - This operation establishes a database connection by logging to the specified database server and sets the appropriate parameters. An error condition will be sent if the connection fails. It is expected that this operation is invoked only when a thread of database accesses are to be started. Connect/DisConnect for each individual database access is prohibitively expensive.
Arguments: char* userName, char* password, char* serverName
Return Type: GlStatus
Privilege: Public

ConnectionState - This operation returns the current databse connection status.
Arguments:
Return Type: DsTDbConnectionState
Privilege: Public

Disconnect - This operation disconnects an established database connection, resets ths state parameters and cleans up internal data structures if appropriate. It is expected that this operation to be invoked only when an application finishes all the databse access tasks. Connect/DisConnect for each individual database access can be prohibitively expensive.
Arguments:
Return Type: GlStatus
Privilege: Public

DsDbInterface - Constructor to setup default values for state variables.
Arguments:
Return Type: Void
Privilege: Public

Execute - This operation allows caller to send a SQL statement and executes the command on their behalf. The execution status will be returned. It does not differentiate between utility command (no result data, e.g. open db, begin transaction,...) and regular query. The regular query upon return sets up an internal handle and the result can be retrieved through FetchQueryResult() calls.
Arguments: char* SQLCmd
Return Type: DsTDbExecStatus
Privilege: Public

ExecutionStatus - This operation returns the query execution status for the last ExecuteQuery() call.
Arguments:
Return Type: DsTDbExecStatus
Privilege: Public

FetchQueryResult - This operation is another interface to retrieve query result. It allows direct binding from a result row to application variables. The pointers (void*) for those application variables are contained the vector argument, and the binding will directly copy values from a result row to those variable pointers. Note that the caller is responsible for setting the pointers in the vector in the exact same order as what's described in the SQL select statement. The binding occurred here is not responsible for checking the type and length of each variables since no such descriptor information is available at this level.
Arguments: RWTPtrOrderedVector<void*>& result
Return Type: DsTDbExecStatus
Privilege: Public

FetchQueryResult - This operation fetches one row of query result and construct a GlParameterList for it. Since more than one row may be contained in the result, caller is expected to put this call into a loop and the loop tern]minates only when the return status is EXEC_ENDOFRESULT. Note that it allocates memory space for each attribute value field and store data in their native format (whatever is stored in the database). Caller is responsible for cleaning up the allocated memory space in the GlParameterList.
Arguments: GlParameterList& result
Return Type: DsTDbExecStatus
Privilege: Public

ReConnect - This operation reestablishes a databse connection if needed. ReConnect() uses the state information saved internally for the connection.
Arguments:
Return Type: GlStatus
Privilege: Public

VerifyConnection - This operation checks the current connection and reconnects if necessary. This is recommended before each ExecuteQuery() call to make sure the database conenction is alive.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsDbInterface - Destructor to close any database connection if necessary and deallocate any in-memory data structures.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDbInterface class has associations with the following classes:
Class: DsDbAccess
Class: DsDbAttributeToTableVector
Class: DsDbEngine
Class: DsDbGranuleToDbVector

4.3.31 DsDeCoreValid Class

Parent Class: Not Applicable
Public: No Distributed Object: No

Persistent Class: True
Purpose and Description:
This class contains validation information for a core metadata attribute. This class is used to perform the first level of validation of that core metadata attribute. The validation criteria is not based on a specific data type. For example, an attribute might be DSSBeginningDate. The validation criteria at this level will simply ensure that the value for this attribute is a valid date. Further validation is performed by the ValidVector class based on the data type where the value of this attribute would be compared against the valid starting date for this type. It is not necessary that each data type have validation criteria for each attribute if the core validation criteria is sufficient.

Attributes:

myName - The name of the core metadata attribute for which this object has validation criteria.
Data Type: RWCString
Privilege: Private
Default Value:

myType - An enumerated value that identifies the type of this attribute. Valid types include string, date, time, short, long, char, and double.
Data Type: DsTAttributeType
Privilege: Private
Default Value: STRING

myValid - Contains an object that has the criteria for validating this attribute.
Data Type: DsDeValid
Privilege: Private
Default Value:

Operations:

DsDeCoreValid - This constructor reads from the given stream to create an instance of this class.
Arguments: from: istream &
Return Type: Void
Privilege: Public

Validate
Arguments: theName:RWCString&, theValue:RWCString &

~DsDeCoreValid - There is no specific implementation for this destructor.
Arguments:

Return Type: Void
Privilege: Public

Associations:

The DsDeCoreValid class has associations with the following classes:
DsDeCoreValidVector (Aggregation)

4.3.32 DsDeCoreValidVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains a collection of CoreValid objects. This class provides services that operate over the collection. The DsDeCoreValidVector class is derived from a standard Rogue Wave vector class.

Attributes:

None

Operations:

DsDeCoreValidVector - This constructor reads the section of the descriptor that contains core valid entries until the end of that section is detected. Core Valid objects are created and added to the vector.
Arguments: from: istream &
Return Type: Void
Privilege: Public

Find - This service finds a core valid entry with the given name. The implementation of this method is inherent in the Rogue Wave class that this is derived from.
Arguments: myName: RWCString
Return Type: DsDeCoreValid *
Privilege: Public

Validate - This method takes a stream containing P=V metadata. It looks at each entry one at a time and finds the associated CoreValid object in its vector. Then it uses the validate service of that object. The GlStatus parameter is updated with errors during the process.
Arguments: myCoreMetadata:istream &, GlStatus &

Return Type: RWBoolean
Privilege: Public

~DsDeCoreValids
Arguments:

Associations:

The DsDeCoreValidVector class has associations with the following classes:
Class: DsDeESDTDescriptor hascoremetadatavalidsfor - The CoreValidVector class contains metadata validation criteria for many data type level ESDT Descriptors. The purpose of this relationship is to allow some validation that is the same for all data types to be shared, thus reducing duplication of information within the descriptors.

4.3.33 DsDeDD Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains Data Dictionary information that are specific to this data type. This information is exported to the Data Management Subsystem.

Attributes:

myDescription - Detailed information that is exported to the Data Management Subsystem which describes contents, formats, valid values, etc.
Data Type: RWCString
Privilege: Private
Default Value: NULL

myName - The name of this data dictionary entry.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

DsDeDD - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.

Arguments: from: istream &
Return Type: Void
Privilege: Public

ExportDD - This service is used to export this data dictionary entry to the Data Management subsystem.
Arguments:
Return Type: GlStatus
Privilege: Public

GetName - This service returns the value in the myName private attribute.
Arguments:
Return Type: RWCString &
Privilege: Public

~DsDeDD - The destructor for this object has no special implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeDD class has associations with the following classes:
DsDeDDVector (Aggregation)

4.3.34 DsDeDDVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains a collection of Data Dictionary objects that are specific to this descriptors type. This class provides services that operate over the collection. The DsDeDDVector class is derived from a standard Rogue Wave vector class.

Attributes:

None

Operations:

DsDeDDVector - This constructor reads the section of the descriptor that contains Data Dictionary information until the end of that section is detected. DsDeDD objects are created and added to the vector.
Arguments: from: istream &
Return Type: Void
Privilege: Public

ExportDD - This service iterates over all of the objects in its contents and exports each one to the Data Management Subsystem.
Arguments:
Return Type: GlStatus
Privilege: Public

Find - This service finds a DsDeDD instance that has the given name. A pointer to this instance is returned. If a DsDeDD object with this name is not found, then the pointer is NULL.
Arguments: theName: RWCString &
Return Type: DsDeDD
Privilege: Public

~DsDeDDVector - The destructor destroys all of the objects in its contents and then destroys itself.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeDDVector class has associations with the following classes:
DsDeESDTDescriptor (Aggregation)

4.3.35 DsDeESDTDescriptor Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class: True
Purpose and Description:
To describe the content, structure and behavior of an ESDT. The schema provides a software description of each ESDT that a data server provides. The description includes the structure and services available for each ESDT.

Attributes:

myAdvertisedServices - Services that the associated ESDT provides that are advertised.
Data Type: DsDeServiceVector
Privilege: Private
Default Value:

myCoreMetadataConfiguration -
Data Type: DsDeMetadataDefVector
Privilege: Private
Default Value:

myDataDictionaryInfo - The description of the ESDT that this ESDT descriptor defines.
Data Type: DsDeDDVector
Privilege: Private
Default Value:

myEvents - Services of this specific ESDT that can have subscriptions against their invokation. For example, the insert service may be subscribable and the browse service may not be subscribable for the CER07 ESDT.
Data Type: DsDeEventVector
Privilege: Private
Default Value:

myProductMetadataConfiguration - Contains the name, validation information, and type for product specific metadata attributes. Also contains a mapping of these attributes to inventory attributes.
Data Type: DsDeMetadataDefVector
Privilege: Private
Default Value:

myScienceParameters - Specification of the parameters within the data that each granule of this type has. This includes information related to the parameter names, their types and their locations in the granule.
Data Type: DsDeScienceParameterVector
Privilege: Private
Default Value:

myStaticMetadata -
Data Type: DsDeStaticMetadataVector
Privilege: Private
Default Value:

myStatus -
Data Type: GlStatus
Privilege: Private
Default Value:

myType - The specific data type that this ESDT Descriptor defines.
Data Type: DsGeTypeID
Privilege: Private
Default Value:

myValids - Valid values for each parameter that will potentially be validated.
Data Type: RWVector<DsDeValid>
Privilege: Private
Default Value:

ourCoreValids -
Data Type: DsDeCoreValidVector
Privilege: Private
Default Value:

Operations:

ConvertToPlist - This service converts the given metadata entries to a parameter list. The given parameter list is filled in with this information.
Arguments: theMD:istream &, theList:GlParameterList &
Return Type: void
Privilege: Public

DsDeESDTDescriptor - The constructor for the descriptor. The TypeID argument must specify the name of the ESDT and optionally its version. If the version is empty, then the most recent version will be used.
Arguments: theType: DsGeTypeID &
Return Type: Void
Privilege: Public

Externalize - This service is used to create an external representation of the ESDT Descriptor. This is most likely an ASCII file.
Arguments: ostream
Return Type: RWBoolean
Privilege: Public

GetCollectionGroup - This service creates a parameter list containing the static metadata for this data type. The return value indicates whether the service completed successfully

or not. If it did not, then the GlStatus parameter will reflect the problems/errors.
Arguments: staticMD: GlParameterList&, status:GlStatus &
Return Type: RWBoolean
Privilege: Public

GetMCF - This service streams the Metadata Configuration information to the given ostream. This is done by telling the MetadataDefVector to externalize itself to the stream.
Arguments: theMCF: ostream &
Return Type: void
Privilege: Public

GetParameter - This service is used to get the name of a well-known parameter. The given enumerated type can be FILENAME or CHECKSUM.
Arguments: name: DsTpname
Return Type: RWCString
Privilege: Public

GetQueryableParameters - This service is used to get the list of attributes that are queryable for the data type. This is accomplished by getting the set of queryable parameters from the MetadataDefVector.
Arguments: theParameters: GlParameterList &
Return Type: void
Privilege: Public

Initialize - This service performs the initialization of this data type. This includes creating and registering events via the EventVector and creating and registering advertisements via the ServiceVector. The expectation is that this service is only used upon startup or configuration of a new data type.
Arguments:
Return Type: RWBoolean
Privilege: Public

Internalize - This service is used to add a new descriptor from a file in support of the function of adding new data types. The file must have the appropriate format.
Arguments: theDef: istream &
Return Type: GlStatus
Privilege: Public

Validate - This service is used to validate a set of P=V metadata. This is done by a 2 step validation process. First, the core metadata validation is done to check that the given metadata at least meets that criteria. Then a validation that is specific to this data type is done where the validation criteria is more restrictive than the core validation.
Arguments: MetadataFile, GlStatus &
Return Type: RWBoolean
Privilege: Public

PDL:

DsDeESDTDescriptor::Validate(istream &theMetadata, GlStatus &theStatus)

This operation takes the given metadata which is in a P=V format and validates that 1)all attributes are valid and 2)all required attributes are present. The GlStatus object is filled with any problems

```
SEQUENCE
  Validate theMetadata against the DsDeCoreValidVector
  Validate theMetadata against the DsDeValidVector
  Determine if all mandatory values are in theMetadata
  IF any of the above returned FALSE it means validation failed
    RETURN FALSE
  ELSE
    RETURN TRUE
END SEQUENCE
```

Validate - This service validates the given Command by finding the DsDeService vector that matches the service in the command and then asking the service to validate the parameters within the command. The return value indicates whether the command is valid or not. If the command is invalid, the given GlStatus parameter indicates the reason that the command failed validation.
Arguments: theCmd: DsSrCommand, status: GlStatus &
Return Type: RWBoolean
Privilege: Public

Withdraw - This service is used to remove this data type from the set of configured data types for this DSS. This is the opposite of the initialization for an ESDT Descriptor. The advertisements are withdrawn and the events are unregistered.
Arguments:
Return Type: RWBoolean
Privilege: Public

~DsDeESDTDescriptor - The destructor for this class has no specific implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeESDTDescriptor class has associations with the following classes:

Class: DsGeESDT

Class: DsGeESDTEventTable creates - The ESDTDescriptor creates the ESDTEventTable upon startup. This table has events that the ESDT instances will notice and report their occurrence so that subscriptions can be fulfilled. The ESDTDescriptor also uses the ESDTEventTable when shutdown occurs. The events must be located in the table and unregistered.

Class: DsDeCoreValidVector hascoremetadatavalidsfor - The CoreValidVector class contains metadata validation criteria for many data type level ESDT Descriptors. The purpose of this relationship is to allow some validation that is the same for all data types to be shared, thus reducing duplication of information within the descriptors.

Class: DsSbEvent registers - The ESDTDescriptor creates and registeres events with the subscription server. These events are one of the mechanisms used for triggering subscriptions.

DsDeESDTDescriptorSet (Aggregation)

4.3.36 DsDeESDTDescriptorSet Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Persistent Class:

Purpose and Description:

To provide a mechanism to contain multiple schema and provide a single reference to these schema and their services. The SchemaSet is the collection of all the individual schemas that a data server contains.

Attributes:

None

Operations:

Add - This service is used to add a new ESDT descriptor to this set of ESDT Descriptors.

Arguments: theName: RWCString, theVersion: RWCString, theDef: istream &

Return Type: GlStatus

Privilege: Public

DsDeESDTDescriptorSet

Arguments:

Return Type: Void
Privilege: Public

Externalize - This service is used to externalize each individual ESDT descriptor in this set of ESDT descriptors.
Arguments:
Return Type: ostream &
Privilege: Public

Initialize - This service is used to initialize all of the ESDT Descriptors that are in this set. Initialization is performed during startup of the DSS.
Arguments:
Return Type: GlStatus
Privilege: Public

Remove - This service removes the given data type from the set of ESDTs that this DSS is configured for.
Arguments: theType: DsGeTypeID
Return Type: Void
Privilege: Public

Replace - This service provides the capability to replace an existing ESDT Descriptor with a new ESDT Descriptor.
Arguments: theType:DsGeTypeID, newDefinition:istream &
Return Type: GlStatus
Privilege: Public

~DsDeESDTDescriptorSet - The destructor for this class destroys each of its members and then destroys itself.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeESDTDescriptorSet class has associations with the following classes:
None

4.3.37 DsDeEvent Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Purpose and Description:

This class contains events that any ESDT instance of this type detects the occurrence of and provides notification of that occurrence.

Attributes:

myDescription - This is a description of the event that instances of this data type will detect. For example, the description of the INSERT event for the CER02 data type might indicate that the INSERT event occurs whenever a CER02 granule has been successfully archived in this DSS. The description also would include information about what information is made available when the event occurs. For example, in the case of INSERT, the UR of the new granule would be provided.
Data Type: RWCString
Privilege: Private
Default Value:

myEventID - The identifier for this Event. This identifier along with the event name uniquely identify an Event object.
Data Type: RWCString
Privilege: Private
Default Value:

myEventName - The name of the event. This name must be unique within the event id. For example, there can only be one event named INSERT for the CER02 eventID. However, there can be an INSERT for each unique eventID (i.e. CER02V1.0, CER02V1.1, LIS01, etc.)
Data Type: RWCString
Privilege: Private
Default Value:

myReturnInformation - This GlParameterList contains the names and types of what data will be provided when the event occurs. For example, an INSERT event's myReturnInformation would include a GlParameter named UR that is of type RWCString. This says that when the event occurs, the ESDT will provide a parameter list with a GlParameter named UR that is of type RWCString and has a value of whatever the new UR is for this new granule.
Data Type: GlParameterList
Privilege: Private
Default Value:

Operations:

DsDeEvent - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from: istream &
Return Type: Void
Privilege: Public

GetDescription - Returns the information in the myDescription attribute.
Arguments:
Return Type: RWCString &
Privilege: Public

GetName - Returns the information in the myEventName attribute.
Arguments:
Return Type: RWCString &
Privilege: Public

Register - This service creates a DsSbEvent which gets registered as a subscribable event with the subscription server.
Arguments:
Return Type: GlStatus
Privilege: Public

Unregister - This service obtains the DsSbEvent associated with this name and ID and unregisters it as a subscribable event with the subscription server.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsDeEvent - There is no special implementation for the destructor.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeEvent class has associations with the following classes:
DsDeEventVector (Aggregation)

4.3.38 DsDeEventVector Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Purpose and Description:

This class contains a collection of DsDeEvent objects that are specific to this descriptors type. This class provides services that operate over the collection. The DsDeEventVector class is derived from a standard Rogue Wave vector class.

Attributes:

None

Operations:

DsDeEventVector - This constructor reads the section of the descriptor that contains event entries until the end of that section is detected. Event objects are created and added to the vector.
Arguments: from: istream &
Return Type: Void
Privilege: Public

FindEvent - This service searches for an event of the given name in its contents. A pointer to the event is returned. If the event is not found, the pointer is NULL.
Arguments: theName: RWCString
Return Type: DsDeEvent *
Privilege: Public

Register - This service iterates over the members in its contents telling each one of them to register themselves with the subscription server.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsDeEventVector - The destructor for this class destroys each of the objects in its contents. The events are still known to the subscription server until the unregister service for an event is invoked.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeEventVector class has associations with the following classes:
DsDeESDTDescriptor (Aggregation)

4.3.39 DsDeMathOp Class

Parent Class: DsDeValid
Public: No Distributed Object: No
Purpose and Description:
This class has validation criteria that is mathematically based. This includes mathematical functions such as "LT", "GT", "EQ", etc. The equivalent mathematical functions are used to validate whether a given value satisfies the criteria.

Attributes:

myDomain - This attribute contains the domain for the mathematical operation. This domain is used to evaluate the mathematical expression and determine whether a given value meets the criteria. For example, the domain might be "CERES" and the MathOperation might be "EQ".
Data Type: T
Privilege: Private
Default Value:

myMathOperation - This attribute contains the representation of the mathematical operation. Valid math operations are "GT", "LT", "EQ", "GE", "LE", and "NE".
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

DsDeMath - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from: istream &
Return Type: Void
Privilege: Public

~DsDeMath - The destructor for this class has no special implementation.
Arguments:

Return Type: Void
Privilege: Public

Associations:

The DsDeMathOp class has associations with the following classes:
None

4.3.40 DsDeMetadataDef Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains configuration information about a metadata entry. This class is used for core as well as product specific metadata. DsDeMetadataDef entries are used to build the metadata configuration file (MCF).

Attributes:

myAttributeName - The name of the metadata attribute as defined in the Core Metadata Model. Valid names/attributes are either core or product specific.
Data Type: RWCString
Privilege: Private
Default Value:

myDataLocation - This attribute indicates the location where this attribute is expected to be found when the metadata is being generated. This location is used by the processing subsystem to flag whether the PGE generates the value, whether it is in the process control file (PCF), etc.
Data Type: RWCString
Privilege: Private
Default Value:

myMandatoryFlag - This flag indicates whether or not this metadata attribute is mandatory.
Data Type: RWBoolean
Privilege: Private
Default Value: False

myType - This indicates the type of this metadata attribute. The valid values for this attribute are enumerated with the default being a STRING.
Data Type: DsTAttributeType
Privilege: Private
Default Value: STRING

myValue
Data Type: RWCString
Privilege: Private
Default Value: NULL

Operations:

DsDeMetadataDef - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from: istream &
Return Type: Void
Privilege: Public

Externalize - This service streams this metadata definition instance to the given ostream. The format of this is a well-defined ODL syntax that is agreed upon between the DSS, Ingest, and Processing. This service is used in creation of the MCF for these external customers.
Arguments: to: ostream &
Return Type: GlStatus
Privilege: Public

Parameterize - This service creates a Parameter whose name is myAttributeName, whose type is myType, and whose value is myValue.
Arguments:
Return Type: GlParameter
Privilege: Public

~DsDeMetadataDef - The destructor for this class has no special implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeMetadataDef class has associations with the following classes:
DsDeMetadataDefVector (Aggregation)

4.3.41 DsDeMetadataDefVector Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Purpose and Description:

This class contains a collection of Metadata definition objects that is specific to this descriptors type. This class provides services that operate over the collection. The DsDeMetadataDefVector class is derived from a standard Rogue Wave vector class.

Attributes:

None

Operations:

DsDeMetadataDefVector - This constructor reads the section of the descriptor that contains metadata defintion entries until the end of that section is detected. Metadata definition objects are created and added to the vector.
Arguments: from: istream &
Return Type: Void
Privilege: Public

Externalize - This service generates the MCF information for this metadata vector by creating the appropriate section headers and writing them to the ostream and then telling each one of its members to externalize itself to that same stream. Then it creates the section footers in the same ostream to complete this service.
Arguments: theMCF: ostream &
Return Type: RWBoolean
Privilege: Public

FindEntry - This service finds a MetadataDef entry that has the same name as the given string. The return value indicates whether the entry was found or not. If the entry was found, the given MetadataDef reference will point to it.
Arguments: theName: RWCString, theEntry: DsDeMetadataDef &
Return Type: RWBoolean
Privilege: Public

GetQueryableParameters - This service fills in the names, types and values (if applicable) of each of the metadata definition objects in its contents into the given

parameter list. This is done by using the parameterize service of the metadata definition objects.
Arguments: theParameters:GlParameterList &
Return Type: RWBoolean
Privilege: Public

HasMandatory - This service determines whether the given stream containing P=V metadata has all of the metadata parameters that are defined as being mandatory. If the return value indicates that all of the mandatory parameters are not present, then the given GlStatus object provides details about which parameters were missing.
Arguments: theMet: istream &, status: GlStatus
Return Type: RWBoolean
Privilege: Public

Parameterize - This service parameterizes each of the objects in its contents into the given parameterList.
Arguments: theParms:GlParameterList &
Return Type: Void
Privilege: Public

~DsDeMetadataDefVector - This destructor destroys all of the objects in its contents and then destroys itself.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeMetadataDefVector class has associations with the following classes:
DsDeESDTDescriptor (Aggregation)

4.3.42 DsDeRange Class

Parent Class: DsDeValid
Public: No Distributed Object: No
Purpose and Description:
This class has validation criteria that specifies a range of values in which any valid entry must fall. This class is templatized so that it can operate over any type.

Attributes:

myHigh - This is the upper limit for the range. Any value that is greater than this value is outside of the valid range.
Data Type: T
Privilege: Private
Default Value:

myLow - This is the lower bound for the range. Any value that is less than this value is outside of the valid range.
Data Type: T
Privilege: Private
Default Value:

Operations:

DsDeRange - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from: istream &
Return Type: Void
Privilege: Public

~DsDeRange
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeRange class has associations with the following classes:
None

4.3.43 DsDeScienceParameter Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains the science parameters of this data type. These parameters are features of the internal format of the science data.

Attributes:

myDescription - This attribute provides a detailed information describing this science parameter, maybe its meaning and units. This information is whatever the data provider deems is a useful description for this science parameter.
Data Type: RWCString
Privilege: Private
Default Value:

myInternalName - This is the name within the science data information that this parameter is referred as. This allows there to be a name that the user community sees and can use to identify the parameter and a name that the DSS uses. An example of an internal name for "Sea Surface Temperature" might be "SST".
Data Type: RWCString
Privilege: Private
Default Value:

myName - The name of a science parameter within this data types structure. An example of this might be "Sea Surface Temperature".
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

DsDeScienceParameter - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from: istream &
Return Type: Void
Privilege: Public

GetDescription - This service returns the myDescription private attribute of this science parameter.
Arguments:
Return Type: RWCString &
Privilege: Public

GetInternalName - This service returns the internal name of this parameter.
Arguments:
Return Type: RWCString &
Privilege: Public

~DsDeScienceParameter - The destructor for this class has no special implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeScienceParameter class has associations with the following classes:
DsDeScienceParameterVector (Aggregation)

4.3.44 DsDeScienceParameterVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains a collection of ScienceParameter objects that are specific to this descriptors type. This class provides services that operate over the collection. The DsDeScienceParameterVector class is derived from a standard RogueWave vector class.

Attributes:

None

Operations:

DsDeScienceParameterVector - This constructor reads the section of the descriptor that contains Science Parameters until the end of that section is detected. DsDeScienceParameter objects are created and added to the vector.
Arguments: from: istream &
Return Type: Void
Privilege: Public

FindScienceParameter - This service finds a science parameter whose name matches the given name. A pointer to the science parameter is returned. If the science parameter is not found, the pointer is NULL.
Arguments: theName: RWCString
Return Type: DsDeScienceParameter *
Privilege: Public

~DsDeScienceParameterVector
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeScienceParameterVector class has associations with the following classes:
DsDeESDTDescriptor (Aggregation)

4.3.45 DsDeSeries Class

Parent Class: DsDeValid
Public: No Distributed Object: No
Purpose and Description:

Attributes:

myDomainValues - This is a vector that contains an explicit list of valid values for this attribute.
Data Type: RWVector<T>
Privilege: Private
Default Value:

Operations:

DsDeSeries - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from: istream &
Return Type: Void
Privilege: Public

~DsDeSeries - The destructor for this class has no special implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeSeries class has associations with the following classes:
None

4.3.46 DsDeService Class

Parent Class: Not Applicable

Attributes:

myDescription - This attribute provides a description of the Service and its parameters and whether they are required or not. In addition, the return values are described.
Data Type: RWCString
Privilege: Private
Default Value:

myName - The name of a service that this type provides. An example of this is INSERT.
Data Type: RWCString
Privilege: Private
Default Value:

myParameterList - This parameterList contains the Parameters that are valid for this service. The parameterList parameter names and their types but no values. An example of a parameter for the INSERT service might have the name "METADATAFILE" and the type GlStringP.
Data Type: GlParameterList
Privilege: Private
Default Value:

myRequiredParameters - This vector contains the names of parameters that are required for invocation of this Service. These names are a subset of the named parameters in myParameterList.
Data Type: RWVector<RWCString>
Privilege: Private
Default Value:

Operations:

Advertise - Advertises this DsDeService so that it is publicly available. The advertisement includes all of the information necessary for a client to create a command that this DSS can fulfill.
Arguments:
Return Type: RWBoolean
Privilege: Public

DsDeService - This constructor reads from the given stream to construct itself.
Arguments: from: istream&
Return Type: Void
Privilege: Public

Validate - Validate takes a GlParameterList as input and validates it against the required and valid parameters for this DsDeService. If there is a problem in the validation, the provided GlStatus object will contain the nature of the problem
Arguments: parms:GlParameterList &, result:GlStatus &
Return Type: RWBoolean
Privilege: Public

Withdraw - This service withdraws an advertisement indicating that this previously advertised service is no longer publicly available.
Arguments:
Return Type: RWBoolean
Privilege: Public

~DsDeService - The destructor for this service has no special functionality.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeService class has associations with the following classes:
DsDeServiceVector (Aggregation)

4.3.47 DsDeServiceVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No

Purpose and Description:
This vector contains DsDeService objects. It is responsible for creating each and for implementing services that operate over the collection of them.

Attributes:

None

Operations:

Advertise - This service iterates over all of its DsDeService objects and tells each one of them to advertise themselves.
Arguments:
Return Type: RWBoolean
Privilege: Public

DsDeServiceVector - This service reads from the given istream and creates the DsDeService objects. It is responsible for noticing when the end of the service information is reached.
Arguments: from:istream &
Return Type: Void
Privilege: Public

Find - This service locates the service in its contents that has the same name as the given argument. A pointer to a DsDeService object is returned. If the service is not found, this pointer is NULL.
Arguments: svc:RWCString&
Return Type: DsDeService*
Privilege: Public

Withdraw - This service iterates over all of the DsDeService objects in this vector and invokes the withdraw service.
Arguments:
Return Type: RWBoolean
Privilege: Public

~DsDeServiceVector - This destructor invokes the destructor for each of the objects in its contents and then destroys itself.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeServiceVector class has associations with the following classes:
DsDeESDTDescriptor (Aggregation)

4.3.48 DsDeStaticMetadata Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains metadata name, type, and value for each attribute that is static for the collection of all granules of this type.

Attributes:

myName - The name of this static metadata attribute.
Data Type: RWCString
Privilege: Private
Default Value:

myType - This indicates the type of this metadata attribute. The valid values for this attribute are enumerated with the default being a STRING.
Data Type: DsTAttributeType
Privilege: Private
Default Value: STRING

myValue - This attribute contains the value of this static metadata entry. This value is by definition the same for all granules of this data type.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

DsDeStaticMetadata - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from: istream &
Return Type: Void
Privilege: Public

Externalize - This service is used to populate the given parameterlist with the contents of this static metadata instance. If necessary, the given GlStatus is updated to reflect problems/errors.
Arguments: to: GlParameterList &, status: GlStatus &
Return Type: RWBoolean
Privilege: Public

GetValue - This service returns a GlParameter containing the value of this static metadata attribute.
Arguments:
Return Type: GlParameter &
Privilege: Public

~DsDeStaticMetadata - The destructor for this class has no specific implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeStaticMetadata class has associations with the following classes:
DsDeStaticMetadataVector (Aggregation)

4.3.49 DsDeStaticMetadataVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains a collection of Static Metadata objects that are specific to this descriptors type. This class provides services that operate over the collection. The DsDeStaticMetadataVector class is derived from a standard Rogue Wave vector class.

Attributes:

None

Operations:

DsDeStaticMetadataVector - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.

Arguments: from:istream &
Return Type: Void
Privilege: Public

Externalize - This service is used to populate the given parameter list with the contents of each StaticMetadata object that is in this vectors contents. In its implementation, this vector simply passes this request to each of its members.
Arguments: to:GlParameterList &, status: GlStatus&
Return Type: RWBoolean
Privilege: Public

FindEntry - This service finds a Static MetadataDef entry that has the same name as the given string. The return value indicates whether the entry was found or not. If the entry was found, the given MetadataDef reference will point to it.
Arguments: theName: RWCString
Return Type: DsDeStaticMetadata *
Privilege: Public

~DsDeStaticMetadataVector - The destructor for this class first calls the destructor of each of its members and then deletes itself.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeStaticMetadataVector class has associations with the following classes:
DsDeESDTDescriptor (Aggregation)

4.3.50 DsDeValid Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains the validation criteria for a metadata attribute. This is an abstract base class. All of the classes derived from this class must implement the isValid method. This class is templatized based on the type of attribute being validated.

Attributes:

myName - The name of the attribute that this object has validation criteria for.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

DsDeValid - This constructor reads from the given stream to construct itself. The stream has well defined attributes in an ODL format that this object knows how to interpret.
Arguments: from:istream &
Return Type: Void
Privilege: Public

IsValid - This service is used to validate the given value. Because this is an abstract base class, each of the subclasses must implement this service. The return of this service indicates whether the value fell within the validation criteria.
Arguments: theValue: T&
Return Type: RWBoolean
Privilege: Public

~DsDeValid - The destructor for this class has no special implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeValid class has associations with the following classes:
DsDeValidVector (Aggregation)

4.3.51 DsDeValidVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class contains a collection of DsDeValid objects that are specific to this descriptors type. This class provides services that operate over the collection. The DsDeValidVector class is derived from a standard Rogue Wave vector class.

Attributes:

None

Operations:

DsDeValidVector - This constructor reads the section of the descriptor that contains Validation criteria entries until the end of that section is detected. DsDeValid objects are created and added to the vector.
Arguments: from:istream &
Return Type: Void
Privilege: Public

Find - This service finds a DsDeValid object in the vector that has the given name. If there is no match, a null pointer is returned. Otherwise a pointer to the object is returned.
Arguments: what: RWCString &
Return Type: DsDeValid *
Privilege: Public

Validate - This service is used to validate the given metadata file containing P=V metadata. This is done by obtaining a metadata entry from the file, finding its validation criteria entry in the vector, and then asking this entry if the value is valid.
Arguments: mdfile:istream &
Return Type: GlStatus
Privilege: Public
PDL:DsDeValidVector::Validate(istream &theMetadata, GlStatus &theStatus)

This operation takes the given metadata which is in a P=V format and validates that the attributes are valid. Any problems are added to theStatus, but validation continues until all attributes have been validated.

```
SEQUENCE
  Set ReturnStatus to SUCCESS
  DO WHILE (there are more entries in theMetadata)
Get next entry from theMetadata
Find matching DsDeValid entry
IF not found THEN
  Add error to status
  Set ReturnStatus to FAILURE
ELSE
  check value against DsDeValid using IsValid service
  IF not valid THEN
    Set ReturnStatus to FAILURE
```

```
            ENDIF
          ENDIF
      END DO WHILE
      RETURN ReturnStatus
    END SEQUENCE
```

~DsDeValidVector
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDeValidVector class has associations with the following classes:
DsDeESDTDescriptor (Aggregation)

4.3.52 DsDoReferencePaper Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To provide an interface to services for reference documents. ReferencePapers are documents that may be used in support of understanding more about a particular science data product.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsDoReferencePaper class has associations with the following classes:

Class: DsGeScienceData isdescribedby - Each science data object class can be described by zero or more instances of a reference paper. Note that this relationship is at the class level for the science data object. For example, a set of reference papers describes all CER03 data objects. This means that each instance of a CER03 data object is described by the same set of reference papers.

4.3.53 DsFactory Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Purpose and Description:

An object of this class is used to establish all client connections to a data server. When a "collector" object is created by the client, it finds the DsFactory object on the desired data server (there is exactly one per data server) and asks it to create a "connection" object on the server to correspond with it. The DsFactory will create this server-side object and return to the "collector" a reference to it, that the collector object can use to perform requests.

Attributes:

None

Operations:

DsFactory - Constructs a data server factory.

Arguments:
Return Type: Void
Privilege: Public

MakeConnection - Used to produce a new connection for the given profile.
Arguments: MSS_UserProfile
Return Type: DsSrConnection*
Privilege: Public

MakeSession - Used to produce a new session for the given profile and connection ID.
Arguments: MSS_UserProfile, DsESrConnectionID = 0
Return Type: DsSrSession*
Privilege: Public

~DsFactory - Destroys this data server factory.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsFactory class has associations with the following classes:
None

4.3.54 DsGeBrowseProduct Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To provide a standard access to data that will be used to aid scientists in assessing their interest in data. The Browse product class is a specialization of the ESDT class. It will provide the services that will allow manipulation of the browse data.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsGeBrowseProduct class has associations with the following classes:
Class: DsGeScienceData has - Each instance of a science data object has zero or more browse products associated with it. A browse product is a representation of the associated science data that can be used to determine whether a science data product should be ordered. Each browse product can represent zero or more science data objects.

4.3.55 DsGeDynamicLibrary Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Purpose and Description:

This class implements the loading and unloading of a dynamically linked library. The reason this class is in the General ESDT class category is that this class is only being used in support of dynamic ESDT configurations. However, anyone who has a need for supporting dynamically linked libraries could use this class.

Attributes:

myHandle - A handle to the library. This is the return value from the OS command to open the library. This handle is used by all other services that operate on the library.

Data Type: void *

Privilege: Private

Default Value:

Operations:

DsGeDynamicLibrary - Constructor for dynamic library doesn't do too much. All of the work is done with the load, unload, and get symbol functions.

Arguments:

Return Type: Void

Privilege: Public

GetSymbol - This service is used to bind to the given symbol within the library. A pointer to the symbol is returned. This service is used by the derived ESDTDynamicLibrary class to obtain the function pointer for the newESDT service.

Arguments: theSymbol:RWCString

Return Type: void *

Privilege: Public

LoadLibrary - Service that opens the library and loads it for further use. The handle for the library is established during this process.

Arguments: libName: RWCString

Return Type: RWBoolean

Privilege: Public

Unload - This service unloads a previously loaded dynamic linked library and frees the associated address space.

Arguments:

Return Type: Void
Privilege: Public

~DsGeDynamicLibrary - The destructor for this class unloads the dynamic linked library that was previously loaded.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsGeDynamicLibrary class has associations with the following classes:
None

4.3.56 DsGeECSDataProduct Class

Parent Class: DsGeScienceData
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To provide a general interface for services provided by ECS Data Product objects. ECSDataProducts are the ESDTs that are science data objects generated by ECS.

Attributes:

myAlgorithmUR - UR of the delivered algorithm package that was used to generate this ECS data product.
Data Type: GlUR
Privilege: Private
Default Value:

myPGE - UR of the science software delivery that was used to generate this ECS data product.
Data Type: GlUR
Privilege: Private
Default Value:

mySSAP - The reference to the Delivered Algorithm Package (DAP) that was used to generate this ECS data product.
Data Type: GlUR

Privilege: Private
Default Value:

mySummaryList - Reference to the list of URs that contain summary products for this ECS data product.
Data Type: GlUR
Privilege: Private
Default Value:

Operations:

GetPGEInfo - This service is used to get information about the PGE that was used to generate this ECS data product.
Arguments:
Return Type: GlUR
Privilege: Public

GetSSAP - This service obtains the science software archive package that was used to generate this ECS data product.
Arguments:
Return Type: GlUR
Privilege: Public

GetSummaryStats - This service is used to obtain the summary statistics for this ECS data product.
Arguments:
Return Type: GlUR
Privilege: Public

HasSummaryStats - This service is used to determine whether or not this ECS data product has summary statistics associated with it.
Arguments:
Return Type: RWBoolean
Privilege: Public

Subsample - This service provides the ability to obtain a representation of the ECS Data Product by using a consistent sampling scheme to extract data. This is a virtual function which is defined in the subclasses.
Arguments: pars:GlParamterList
Return Type: GlUR
Privilege: Public

Subset - This service provides the ability to extract a full resolution portion of the ECS data product. It is a virtual function that is defined in a derived class.
Arguments: pars:GlParameterList
Return Type: GlUR
Privilege: Public

Associations:

The DsGeECSDataProduct class has associations with the following classes:
Class: DsNsScienceSoftwareArchivePackage isusedtocreate - An instance of a delivered algorithm package (DAP) is used to create zero or more ECS data products. The DAP contains the software, executables, makefiles, etc. The processing subsystem uses the DAP to generate ECS data products. Each ECS data product was created using one DAP.
Class: DsGeSummaryProduct summarizes - A summary product is an ECS data product that can summarize zero or more ECS data products. Each ECS data product can be summarized by zero or more summary products. For example, an ECS data product can have daily, weekly, and monthly summary products.

4.3.57 DsGeESDT Class

Parent Class: DsGeESDTServiceProvider
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
The Earth Science Data Type (ESDT) is a superclass for the various specializations that represent the specific data types. The ESDT organizes and provides an interface to the external and internal services. Most probably this class will be an abstract base class. If there are instances of this class it will be to support a generic type of ESDT that has no data type services other than get, put, and search.

Attributes:

myArchive - A reference to the archive that this ESDT is stored in.
Data Type:
Privilege: Private
Default Value:

myInterfaceConfigList - The list of interface configurations for the publicly available services of the ESDTs.
Data Type: DsCnESDTGUIConfiguration

Privilege: Private
Default Value:

myMetadata - This is a reference to a metadata object that contains all of the metadata for this instance. This includes core and product specific metadata. This attribute's value gets set by the Fill service. The inspect service uses the metadata object to obtain the values for desired metadata attributes.
Data Type: DsMdMetadata
Privilege: Private
Default Value:

mySize - The size in bytes of this ESDT. This figure is based on the ESDT if it were externalized.
Data Type: size_t
Privilege: Private
Default Value:

myType - Contains the type of the instance of an Earth Science Data Type object.
Data Type: DsGeTypeID
Privilege: Private
Default Value:

myUR - Universal reference for this ESDT. This is either given upon construction, obtained from the given metadata, or generated during an insert.
Data Type: GlUR
Privilege: Private
Default Value:

Operations:

Archive - This service provides the ability to permanently save this ESDT. This is a protected service used by the ESDT when performing the publically advertised insert service.
Arguments:
Return Type: GlStatus
Privilege: Protected

Externalize - This service is used to create an external representation of an ESDT from its internal representation. This typically means creating one or more files. This is a virtual service which is defined in derived classes.
Arguments: toWhere: ostream &
Return Type: GlStatus
Privilege: Protected

Fill - The fill service uses the given metadata object to populate the attributes for this object. For some of the attributes that are populates, the ESDT asks the metadata object for the values. For example, the ESDT maintains the UR and the archive which it obtains from the metadata object during this service.
Arguments: theMD: DsMdMetadata
Return Type: void
Privilege: Public

GetGUIConfiguration - This service is used by the ESDT to obtain the GUI configuration associated with the given service.
Arguments: svc: RWCString
Return Type: GlStatus
Privilege: Protected

GetQueryableParameters - This service fills in the contents of the given parameter list with the names of the metadata attributes for this ESDT. One of the ways that this could be done is through the ESDT descriptor.
Arguments: GlParameterList &
Return Type: RWBoolean
Privilege: Protected

GetServiceList - This service is used by the ESDT to create a list of the services that this ESDT can perform. It is used to perform the IsServiceAvailable function.
Arguments:
Return Type: RWVector<RWCString>
Privilege: Protected

GetSize - This service returns the size of this ESDT. This is the size in bytes that the file(s) would consume if the ESDT were externalized.
Arguments:
Return Type: size_t
Privilege: Protected

Inspect - This service provides the capability to obtain the UR for this ESDT and to get any optionally selected metadata values for this ESDT. The parameter list contains the name of the metadata entries for which the values are desired. The resultant values are filled into the given parameter list and a status is provided. The status indicates whether the call was successful or not. If not, the reason for failure will be returned in the status.
Arguments: whatEntries:GlParameterList &
Return Type: GlStatus
Privilege: Protected

Internalize - This service is used to create an internal representation of an ESDT from an external representation. This is typically a set of one or more files. This is a virtual service which is defined in derived classes.

Arguments: theArgs: GlParameterList
Return Type: GlStatus
Privilege: Protected

Type - This service reports the type of the ESDT that this object is.
Arguments:
Return Type: RWCString
Privilege: Protected

Update - The update service is used to selectively modify portions of the metadata for this ESDT. This service will restricted to users with the appropriate authorization. This is the service that would be used to update QA information about a data object. This method is protected and only accessible through the execute command service.
Arguments: newMDValues: GlParameterList &
Return Type: GlStatus
Privilege: Protected

Validate - This service checks that the metadata values for this ESDT are valid and that the required metadata is present. This service is protected and is used by the ESDT during insertion of a new granule to ensure that the metadata is valid. This service is performed in cooperation with the ESDT descriptor.
Arguments: MetadataFile: RWCString &, Results: GlParameterList &
Return Type: GlStatus
Privilege: Protected

Associations:

The DsGeESDT class has associations with the following classes:

Class: DsGeTypeID

Class: DsGeESDTEventTable restoresfrom - Part of an ESDTs definition is that it notices when specific activities occur. Events corresponding to each of these activities are created and saved to an event table when the data server starts up. Later when one of the activities occurs, the ESDT is responsible for restoring the corresponding event from the event table.

Class: DsSbEvent triggers - The ESDT is responsible for triggering events. This is done to indicate that a previously registered action occurred. The event is responsible for performing whatever action(s) are required upon the occurrence of that event. For example, an insert event is registered. When a granule is inserted, that event is triggered. This tells the event that an insert has occurred. The event can ignore this occurrence or notify subscribers to that event or whatever is appropriate.

Class: DsGeESDTWrapper wraps - The ESDT Wrapper is used as a means to access the services of the ESDT. The Wrapper is the recipient of all commands that are destined for an ESDT. These commands are simply passed through to the ESDT.

4.3.58 DsGeESDTConfiguration Class

Parent Class: DsCnConfiguration

Public: No Distributed Object: No

Purpose and Description:

This object contains the configuration for all of the ESDTs for which services exist for this data server. The configuration for each ESDT consists of a descriptor, an event table, and a dynamically linked library. The services that this class provides may be subsumed by a database implementation of the needed information based on DsGeTypeID. In the prototype, this classes persistence is maintained in a file.

Attributes:

myESDTTypes - The ESDTs that this data server is configured for. This set of types does not imply that there exist granules for each type but that there are services for each type.
Data Type: RWTPtrOrderedVector<DsGeTypeID>
Privilege: Private
Default Value:

Operations:

DsGeESDTConfiguration - The constructor takes the name of the file containing all of the configuration information for the ESDTs for this data server.
Arguments: filename:RWCString &
Return Type: Void
Privilege: Public

GetAllDataTypes - Returns the list of all ESDTs that this data server is configured for. This service is used during data server startup to initialize each of the ESDTs in the list.
Arguments:
Return Type: RWTPtrOrderedVector<DsGeTypeID>
Privilege: Public

GetDefinitionFileForType - Returns the name of the ESDT Descriptor file for the given type.
Arguments: theType:DsGeTypeID &
Return Type: RWCString
Privilege: Public

GetESDTToken - Returns a string containing the token within the configuration file for the given TypeID.
Arguments: theType:DsGeTypeID &
Return Type: RWCString
Privilege: Private

GetEventTableForType - Returns the name of the event table file for the given TypeID.
Arguments: theType:DsGeTypeID &
Return Type: RWCString
Privilege: Public

GetLibraryForType - Returns the name of the dynamically linked library that is the implementation of the given TypeID.
Arguments: theType:DsGeTypeID &
Return Type: RWCString
Privilege: Public

~DsGeESDTConfiguration - The destructor for this class has no implementation other than that derived from the base class.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsGeESDTConfiguration class has associations with the following classes:

Class: DsGeESDTServiceProvider isinformedby - The ESDT configuration is used by the ESDT service provider's derived classes to access various portions of the ESDTs configuration. The ESDT Wrapper obtains the DLL information from it and the ESDT obtains the Descriptor and Event Table information from it.

4.3.59 DsGeESDTDynamicLibrary Class

Parent Class: DsGeDynamicLibrary

Public: No Distributed Object: No
Persistent Class: True
Purpose and Description:
This class provides the functionality needed to load dynamically linked libraries for an ESDTs implementation. The specifics of loading the library are inherited from the DsSdDynamicLibrary class. The library must include the implementation for 2 functions that have C entry points. These functions are newESDT which returns a pointer to a new

object of the actual type and deleteESDT which destroys an object that was generated using newESDT.

Attributes:

myDLLFileName - The file name of the dynamic linked library that contains the implementation for this ESDT.
Data Type: RWCString
Privilege: Private
Default Value:

myNewFunction - This attribute is used when obtaining a pointer to the function that performs the new operation for the ESDT in this dynamic library. This means that each dynamic library must have a function called newESDT which returns a pointer to the ESDT base class. This is a C function that has been declared with extern "C" ESDT *newESDT() so that the name does not get mangled when compiled with CC.
Data Type: DsGeESDT *()
Privilege: Private
Default Value:

myTypeID - The ESDT type information which is used to determine which dynamic library to load. Each type has its own dynamic linked library implementation.
Data Type: DsGeTypeID
Privilege: Private
Default Value:

Operations:

DsGeESDTDynamicLibrary - The constructor loads the dynamic linked library for the given TypeID. This requires that the DLL name be obtained based on the TypeID. This will most likely be done using a DBAccess method which encapsulates access to a database to do a find me the DLL name given the TypeID. If this is not the case, the ESDT configuration will provide this information.
Arguments: DsGeTypeID &
Return Type: Void
Privilege: Public

newESDT - This service calls the newESDT service which each ESDT implementation must provide. The newESDT service actually allocates an ESDT of the exact type. A pointer of the base class type which references an object of the exact type is returned by this call.
Arguments:

Return Type: DsGeESDT *
Privilege: Public

~DsGeESDTDynamicLibrary - The destructor for this class which has no implementation beyond its base class destructor.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsGeESDTDynamicLibrary class has associations with the following classes:
Class: DsGeESDTWrapper isloadedby - The ESDT Wrapper uses the ESDT Dynamic Library to load the implementation for the desired data type and to return a pointer to a real object of that type.

4.3.60 DsGeESDTEventTable Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:

Attributes:

None

Operations:

None

Associations:

The DsGeESDTEventTable class has associations with the following classes:
Class: DsGeESDT restoresfrom - Part of an ESDTs definition is that it notices when specific activities occur. Events corresponding to each of these activities are created and saved to an event table when the data server starts up. Later when one of the activities occurs, the ESDT is responsible for restoring the corresponding event from the event table.

4.3.61 DsGeESDTServiceProvider Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This abstract base class is a generalization of the ESDT and ESDT Wrapper classes. It has virtual functions for all of the public services that are available for all ESDTs.

Attributes:

None

Operations:

DsGeESDTServiceProvider - The constructor for this abstract base class. There is really nothing for this service to do.
Arguments:
Return Type: Void
Privilege: Public

ExecuteCommand - The service that provides a generic interface to the ESDTs. This generic interface allows the services that this type can perform to be extended with no changes to the base classes.
Arguments: theCmd: DsSrCommand &
Return Type: GlStatus
Privilege: Public

IsServiceAvailable - This service is implemented by each subclass and is used to determine if a service with the given name is available for this instance.
Arguments: svcName: RWCString &
Return Type: RWBoolean
Privilege: Public

~DsGeESDTServiceProvider - The destructor for this abstract base class needs no implementation.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsGeESDTServiceProvider class has associations with the following classes:

Class: DsGeESDTConfiguration isinformedby - The ESDT configuration is used by the ESDT service provider's derived classes to access various portions of the ESDTs configuration. The ESDT Wrapper obtains the DLL information from it and the ESDT obtains the Descriptor and Event Table information from it.

4.3.62 DsGeESDTWrapper Class

Parent Class: DsGeESDTServiceProvider

Public: No Distributed Object: No

Persistent Class: True

Purpose and Description:

This class provides a wrapper around the ESDT classes. Its main purpose in life is to load the ESDTs implementation and to create an actual ESDT of the proper type. It has a few other functions that are common to all ESDTs.

Attributes:

myDll - This is a pointer to the dynamic library class that holds the implementation of the actual ESDT that this instance wraps.

Data Type: DsSdESDTDynamicLibrary *

Privilege: Private

Default Value: NULL

myESDT - This is a pointer to the actual ESDT that this instance wraps. This base class pointer points to a real ESDT of the type that it is (i.e. CER03, LIS02, etc.).

Data Type: DsGeESDT *

Privilege: Private

Default Value: NULL

Operations:

DsGeESDTWrapper - The constructor for the ESDT wrapper is given the TypeID for the desired data type. It constructs an ESDT Dynamic Library and issues the newESDT function to obtain a pointer to a real ESDT of the actual type.

Arguments: datatype: DsGeTypeID &

Return Type: Void

Privilege: Public

DsGeESDTWrapper - This constructor takes a metadata object and determines the TypeID from that object to create an ESDT of the right type. After the right ESDT has been created, it is given the metadata object to fill in its attributes.
Arguments: metadata: DsMdMetadata
Return Type: Void
Privilege: Public

~DsGeESDTWrapper - The destructor for the wrapper deletes the ESDT object allocated during the creation of the ESDT dynamic library instance and then deletes the dynamic library instance.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsGeESDTWrapper class has associations with the following classes:

Class: DsGeESDTDynamicLibrary isloadedby - The ESDT Wrapper uses the ESDT Dynamic Library to load the implementation for the desired data type and to return a pointer to a real object of that type.

Class: DsGeESDT wraps - The ESDT Wrapper is used as a means to access the services of the ESDT. The Wrapper is the recipient of all commands that are destined for an ESDT. These commands are simply passed through to the ESDT.

4.3.63 DsGeScienceData Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To provide an interface to services for data that directly supports the earth science investigations and queries. ScienceData is an ESDT that directly supports the services provided on science data products as opposed to non-science data products such as documents, product history and the like.

Attributes:

myBrowseList - Reference to the URs that are browse products for this science data product. Alternatively this may be a single UR for a collection which contains the browse products for this science data product.

Data Type: GlUR
Privilege: Private
Default Value:

myProductHistory - Reference to the list of URs for the products that were used to generate this product.
Data Type: GlUR
Privilege: Private
Default Value:

myQAStatistics - Reference to the UR that represents the QA statistics for this data object.
Data Type: GlUR
Privilege: Private
Default Value:

myReferencePapers - A list of URs for the Reference Papers that describe this science data. Alternatively, this may be a UR for a collection that contains multiple reference papers.
Data Type: GlUR
Privilege: Private
Default Value:

Operations:

AddReferencePaper - This service is used to add a reference paper to the set of reference papers that describe this science data.
Arguments: thePaper:GlUR &
Return Type: Void
Privilege: Public

Browse - This service is used to obtain a browse product(s) that is related to a specific instance of a science data object.
Arguments: pars:GlParameterList &
Return Type: Void
Privilege: Public

GetProductionHistory - This service is used to get the production history for this instance of a science data object.
Arguments:
Return Type: GlUR
Privilege: Public

GetQADataStatistics - This service is used to get QA statistics that this science data may have.
Arguments:
Return Type: GlUR
Privilege: Public

GetReferencePapers - This service is used to get the set of reference papers that describe this science data. Note that the set of reference papers describing a class of science data is the same for all instances of the class.
Arguments:
Return Type: GlUR
Privilege: Public

HasBrowse - This service reports on whether this instance of a science data object has at least one browse product associated with it.
Arguments:
Return Type: RWBoolean
Privilege: Public

RemoveReferencePaper - This service is used to delete a reference paper from the list of reference papers that describe this science data. Note that this is not the same as deleting the reference paper.
Arguments:
Return Type: GlStatus
Privilege: Public

Associations:

The DsGeScienceData class has associations with the following classes:

Class: DsNsProductionHistory describesgenerationof - Science data is generated using various algorithms on a variety of host machines and with various input data. Production history is used to describe the specific inputs, processing, and environment related to the generation of the associated science data object. Each instance of a science data object has an instance of a production history.

Class: DsGeBrowseProduct has - Each instance of a science data object has zero or more browse products associated with it. A browse product is a representation of the associated science data that can be used to determine whether a science data product should be ordered. Each browse product can represent zero or more science data objects.

Class: DsNsQAStatistics has - An instance of a science data object may or may not have an instance of QA data statistics. An instance of QA data statistics indicates the quality of the associated science data object.

Class: DsDoReferencePaper isdescribedby - Each science data object class can be described by zero or more instances of a reference paper. Note that this relationship is at

the class level for the science data object. For example, a set of reference papers describes all CER03 data objects. This means that each instance of a CER03 data object is described by the same set of reference papers.

4.3.64 DsGeSummaryProduct Class

Parent Class: DsGeECSDataProduct
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
A summary product is an ECS data product that represents a summary of the information contained in another ECS data product. For example, a monthly product is and ECS data product but it may also be the summary product for a daily product.

Attributes:

myGranuleList - A list of URs that are summarized by this summary product.
Data Type: GlUR
Privilege: Private
Default Value:

Operations:

AddGranule - This service adds the given granule to the list of granules summarized by this instance of a summary product.
Arguments: data:GlUR &
Return Type: Void
Privilege: Public

ListGranules - This service lists the granules that are summarized by this instance of a summary product. The UR that is returned refers to a list containing the URs of these products.
Arguments:
Return Type: GlUR
Privilege: Public

RemoveGranule - This service is used to remove granules from the list of granules summarized by this instance of a summary product.
Arguments:
Return Type: RWBoolean
Privilege: Public

Associations:

The DsGeSummaryProduct class has associations with the following classes:

Class: DsGeECSDataProduct summarizes - A summary product is an ECS data product that can summarize zero or more ECS data products. Each ECS data product can be summarized by zero or more summary products. For example, an ECS data product can have daily, weekly, and monthly summary products.

4.3.65 DsGeTypeID Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Persistent Class: True

Purpose and Description:

This object uniquely identifies each ESDT's type. The type consists of a type name and a version number. Each type-version number pair is assigned a unique code. The set of all TypeIDs is stored persistently in a database.

Attributes:

myCode - A unique number assigned to this type/version combination. Presumably it will be more efficient to use this number instead of the name and version to identify the type of an ESDT.

Data Type: unsigned long

Privilege: Private

Default Value:

myName - The name of this type. This name along with the version number is enough to uniquely identify this type. An example of a type name is CER03.

Data Type: RWCString

Privilege: Private

Default Value:

myVersion - The version number of this ESDT. Version numbers are needed because an ESDT of the same name may have several implementations.

Data Type: RWCString

Privilege: Private

Default Value:

Operations:

DsGeTypeID - This constructor creates a TypeID instance from the given code. This code uniquely identifies a TypeID.
Arguments: RWCString &theCode
Return Type: Void
Privilege: Public

DsGeTypeID - This constructor takes the name of the type and an optional version number which it uses to create a TypeID. If the version number is not given, then the TypeID is created using the highest version for the given name.
Arguments: RWCString &theName, RWCString theVersion=NULL
Return Type: Void
Privilege: Public

GetTypeCode - Returns the code for this TypeID.
Arguments:
Return Type: unsigned long &
Privilege: Public

GetTypeName - Returns the myName attribute which is the type name of this instance.
Arguments:
Return Type: RWCString
Privilege: Public

GetTypeVersion - Returns the string contained in the myVersion attribute. This is the version of this TypeID.
Arguments:
Return Type: RWCString
Privilege: Public

SetTypeCode - This is a private member function used to set the value of the myCode attribute.
Arguments: unsigned long
Return Type: Void
Privilege: Private

SetTypeName - This is a private member function used to set the value of the myName attribute.
Arguments: RWCString &
Return Type: Void
Privilege: Private

SetTypeVersion - This is a private member function used to set the value of the myVersion attribute.
Arguments: RWCString &
Return Type: Void
Privilege: Private

~DsGeTypeID - The destructor for this type has no implementation at this time.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsGeTypeID class has associations with the following classes:
Class: DsGeESDT

4.3.66 DsGvRadar Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
Class represents data derived directly from the TRMM Ground Based Validation Radar (GV) network. This network is supported by a network of rain gauges.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsGvRadar class has associations with the following classes:
None

4.3.67 DsLiLIS Class

Parent Class: DsGeECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class represents data products derived from the TRMM Lightening Imaging Sensor (LIS).

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsLiLIS class has associations with the following classes:
None

4.3.68 DsMdCatalog Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class is used to provide access to the metadata database. It contains a single database connection, thus a separate instance of this class is required for each thread of execution requiring access to the database. This class represents the logical starting point for access to the database. All Insert, Update, Add, Remove, and Delete methods are executed within the bounds of a database transaction, thus consistency is gauranteed. This class interfaces with the database in terms of Databases, Tables and SQL. It relies on the DsDbInterface class for Sybase specific routines.

Attributes:

myResultVector - This vector is used to hold instances of DsMdMetadata objects. These objects are created within the GranuleSearch method and represent the result set from the query.
Data Type: RWPtrOrderedVector<DsMdMetadata>

Privilege: Private
Default Value: null

mySQLQuery - This is a simple character string buffer used to hold the sql command that is derived from the Query request.
Data Type: char *
Privilege: Private
Default Value: null

Operations:

CollectionSearch - This method searches the collection level metadata for the given PVL constraints and returns a vector of DsMdMetadata objects through the vector parameter.
Arguments: char *pvlString, RWTPtrOrderedVector<DsMdMetadata>& collectionMetadata
Return Type: GlStatus
Privilege: Public

DeleteCollectionMetadata - This method removes the collection level metadata object identified by collectionId.
Arguments: DsTMdIdentifier collectionId
Return Type: GlStatus
Privilege: Public

DeleteCollectionMetadata - This method retrieves the collectionId of the named collection and calls the DeleteCollectionMetadata method with the collectionId.
Arguments: char *collectionName
Return Type: GlStatus
Privilege: Public

DeleteGranuleMetadata - This method deletes all the metadata attributes associated with the granuleId parameter.
Arguments: DsTMdIdentifier granuleId
Return Type: GlStatus
Privilege: Public

DsMdCatalog - This constructor simply calls the Initialize method. Initialization is separate so that it can be isolated if necessary to guarantee success of the constructor.
Arguments: void
Return Type: void
Privilege: Public

GetCollectionMetadata - This method retrieves the metadata attributes associated with the collection identified by the collectionId parameter.
Arguments: DsTMdIdentifier collectionId, DsMdMetadata& metadata
Return Type: GlStatus
Privilege: Public

GetCollectionMetadataTh
Arguments: char *collectionName, DsMdMetadata& metadata
Return Type: GlStatus
Privilege: Public

GetDistributionMetadata - This method retrieves the Metadata that is distributed with a granule.
Arguments: DsTMdIdentifier id, DsMdMetadata &metadata
Return Type: GlStatus
Privilege: Public

GetEphemeris - This method retrieves the ephermeris DsMdMetata object associated with the Granule Identifier parameter.
Arguments: DsTMdIdentifier id, DsMdMetadata &metadata
Return Type: Void
Privilege: Public

GetGranuleMetadata - This method retrieves the metadata attributes associated with the given granuleId.
Arguments: DsTMdIdentifier id, DsMdMetadata &metadata
Return Type: GlStatus
Privilege: Public

GetHouseKeeping - This method retrieves the DsMdMetadata object containing the HouseKeeping attributes associated with the Granule Identifier parameter.
Arguments: DsTMdIdentifier id, DsMdMetadata &metadata
Return Type: GlStatus
Privilege: Public

GetProductionHistory - This method retrieves a DsMdMetadata object containing the Production History attributes associated with a granule.
Arguments: DsTMdIdentifier collectionId, DsMdMetadata& metadata
Return Type: GlStatus
Privilege: Public

GetUniqueID - This method retrieves a unique identifier of type DsTMdIdentifier from the database. The identifier is unique within the scope of the Database Server instance.
Arguments: DsTMdIdentifier& id

Return Type: GlStatus
Privilege: Public

GranuleSearch - This method is responsible for parsing and executing database queries for granules. The InvQuery object parameter contains a PVL based Query description. This PVL is parsed, the databases to query are identified, and SQL is created for each database. The SQL statments are exected using the DsDbInterface object. A myResultVector is used to store results of the query. A new DsMdMetadata object is created for each granule in the result set and inserted into the myResultVector. Finally a new RWTPtrOrderedVector is created and myResultVector is copied to it and returned to the calling routine. It is the responsibility of the calling routine to delete the returned vector. Two options are being evaluated for processing the search: Option 1 basically attempts to include all "single-valued" attributes (granuleId, name, boundingCooridinates, etc) in the first SQL statement. The result set is then processed sequentially retrieving "multi-valued" attributes (discipline keywords, geophysicalParameterKeywords, temporalRange, etc.) by granuleId. This option offers the least complex programatic interface but has questionable performance characteristics. Option 2 has two approaches. Both attempt to retrieve as many attributes as possible in each SQL statement (thus minimizing the number of SQL statements). Both approaches also feature a loop that unloads results into one GlParameterList per unique granuleId. The variance is in the approach to unloading the denormalized result sets: Approach 1 assumes a GlParameterList with extentions for handling set logic. Thus all retrieved attributes can be inserted without reqard to redundancy. The GlParameterList set features will handle the duplication of attributes. The performance implications of this approach need to be evaluated. Approach 2 assumes that the unloading functions are responsible for determining when duplicate values are possible and handling them using knowledge of result set context. This approach will have the best performance but will require the most code and maintenance.
Arguments: InvQuery& query, RWPtrOrderedVector<DsMdMetadata>& metadata
Return Type: GlStatus
Privilege: Private

Initialize - This method creates a DsDbConnection object. The method then creates and initializes a DsDbGranuleToDbVector object. Then a DsDbAttributeToTableVector object is created and initialized.
Arguments: void
Return Type: GlStatus
Privilege: Public

InsertCollectionMetadata - This method inserts a new collection level metadata object and returns the assigned id in the collectionId parameter.
Arguments: DsTMdIdentifier collectionId, DsMdMetadata& metadata
Return Type: GlStatus
Privilege: Public

InsertGranuleMetadata - This method processes the vector of DsMdMetadata objects and iteratively calls the single InsertGranuleMetadata method inserting each object.
Arguments: RWTPtrOrderedVector<DsMdMetadata>
Return Type: GlStatus
Privilege: Public

InsertGranuleMetadata - This method assignes a unique identifier to the granule and inserts it's metadata attributes into the database. The identifier is returned in the granuleId parameter.
Arguments: DsMdMetadata& metadata
Return Type: GlStatus
Privilege: Public

InsertGranuleMetadata - This method inserts a DsMdMetadata object. It is unique in the fact that it allows the caller to specify the granuleId.
Arguments: DsTMdIdentifier granuleId, DsMdMetadata& granuleMetadata
Return Type: GlStatus
Privilege: Public

Search
Arguments: GlParameter& searchConstraints, RWTPtrOrderedVector<DsMdMetadata>*, GlStatus&

UpdateCollectionMetadata - This method takes a DsMdMetadata object as a parameter and updates the attributes associated with the collectionId parameter. This method is equivalent to a delete and add in the case of multivalued attributes. An update is performed for single valued attributes.
Arguments: DsTMdIdentifier collectionId, DsMdMetadata& newMetadata
Return Type: GlStatus
Privilege: Public

UpdateCollectionMetadata - This method simply offers a different signature for UpdateCollectionMetadata. The method first determines the collectionId of the named collection and then calls the UpdateCollectionMetadata with the collectionId as a parameter.
Arguments: char *collectionName, DsMdMetadata& newMetadata
Return Type: GlStatus
Privilege: Public

UpdateGranuleMetadata - This method updates the metadata attributes associated with the granuleId parameter. For multivalued attributes this is equivalent to a delete and add, an update is preformed for single valued attributes.
Arguments: DsTMdIdentifier granuleId, DsMdMetadata& newMetadata
Return Type: GlStatus
Privilege: Public

~DsmdCatalog - This is the destructor for the DsMdCatalog object. It disconnects from the database by deleting the DsDbConnection object. It then deletes the DsDbAttributeToTableVector and the DsDbGranuleToDbVector objects
Arguments: void
Return Type: void
Privilege: Public

Associations:

The DsMdCatalog class has associations with the following classes:
Class: DsDbAttributeToTableVector
Class: DsDbGranuleToDbVector
Class: DsDbInterface

4.3.69 DsMdMetadata Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
The DsMdMetadata class representes a metadata object in the data server subsystem. It includes all the metadata attribute information and their lookup operations. The attributes are stored in the format of a GlParameterList internally. It also supports operations like importing and exporting metadata object from/to external sources such as PVL strings. Metadata attribute update is also supported through this class. Note that any updates to the metadata object only occurs in memory for this class. The update operation on the DsMdCatalog has to be involked to make the change persistent.

Attributes:

myId - Object Id for a Metadata object.
Data Type: DsTMdIdentifier
Privilege: Private
Default Value: null

myNonScienceMetadata - The part of the metadata that is internal to the system. It includes bookkeeping information on the scientific data object. (e.g. file archive name and path for a granule)
Data Type: GlParameterList

Privilege: Private
Default Value:

myScienceMetadata - The part of the metadata that is visible to the end users. It is stored as a list of GlParameters containing attribute names and values. Note that a multi-valued attribute is a single entry (GlParameter) in the GlParameterList which points to a GlParameterList for its values.
Data Type: GlParameterList
Privilege: Private
Default Value:

myType - Type of the metadata object. The metadata object can be for collection, granual and document object. It helps to identify the associated type of the object.
Data Type: char*
Privilege: Private
Default Value: null

myUpdatedAttributes - Updated attribute list of the metadata object. This is used for performance enhancement in updating partial objects. It is constructed/expanded during the update operation and used by the DsMdCatalog class to figure out which attributes got updated and issue command to update only those attribute in the database.
Data Type: GlParameterList
Privilege: Private
Default Value:

Operations:

DsMdMetadata - Constructor to set up default values for state variables.
Arguments:
Return Type: Void
Privilege: Public

GetAttribute - This operation provides a way for caller to query any one attribute value for the metadata object. It takes the name of the attribute of interest, and gets the value of the attribute and store the value in the GlParameter. Note that for performance reason, the the GlParameter returned is a shallow copy of the metadata attribute.
Arguments: char* attributeName, GlParameter& attribute
Return Type: GlStatus
Privilege: Public

GetAttributes - This operation allows caller to get a list of attribute values from the metadata object. It takes a list of GlParamters that containing the attribute names and fills rest of the value fields (even types). The values will be in their native format. Note that for performance reason, the values are shallow copies of the GlParameters of the metadata

object, the data value pointers are shared. This is a multi-scalar version of the GetAttribute() operation.
Arguments: GlParameterList& attributes
Return Type: GlStatus
Privilege: Public

GetUpdatedAttributes - This operation returns a list of the metadata attributes that have been updated by user (via either UpdateAttribute or UpdateAttributes). It is useful for the DsMdCatalog class for making the updates persistent in the databse. The DsMdCatalog update operation can take the return list and only issue databse update command on the corresponding attributes.
Arguments: GlParameterList& modifiedAttributes
Return Type: GlStatus
Privilege:

LoadFromExternal - This operation imports the metadata object from its external format (e.g. PVL string) to the in-memory format. It is responsible for parsing the PVL string and storing attributes (in their native format) in the GlParameterList. Multi-valued attribute is represented as a single GlParameter with its value being a GlParameter for the values. This operation is useful for the ingest process.
Arguments: char* externalForm
Return Type: GlStatus
Privilege: Public

MyId - Return the Metadata object Id. (Identity function)
Arguments:
Return Type: DsTMdIdentifier
Privilege: Public

MyType - This operation reurns the type of the metadata object.
Arguments:
Return Type: char*
Privilege: Public

PutAttribute - This operation allows caller to add an attribute to the metadata object. It takes an attribute (represented as a GlParameter) and insert a copy of it to the GlParameterList.
Arguments: const GlParameter& attribute
Return Type: GlStatus
Privilege: Public

PutAttributes - This operation allows caller to add a list of attributes into the metadata object. The input attributes are expected to be in the format of a list of GlParameterList and the operation will create a deep copy of the list and append to its attribute list. This is a multi-scalar version of the PutAttribute() operation.

Arguments: const GlParameterList& attributes
Return Type: GlStatus
Privilege: Public

SaveToExternal - This operation exports the in-memory format of the metadata object to its external format(e.g. PVL string). It is useful for the distribution process.
Arguments: char*& externalForm
Return Type: GlStatus
Privilege: Public

SetMyId - This operation sets the Id for the metadata object. This is useful for applications that need to access the metadata object through OID handles (e.g. UR).
Arguments: DsTMdIdentifier& Id
Return Type: Void
Privilege: Public

SetMyType - This operation sets the type of the object.
Arguments: char* type
Return Type: Void
Privilege: Public

UpdateAttribute - This operation updates the specified attribute in the metadata object. The attribute to be updated is represented in the format of GlParameter containing both the attribute name and its value. Note the update only occurrs in memory, an update operation to the DsMdCatalog is required to make the change persistent.
Arguments: const GlParameter& attribute
Return Type: GlStatus
Privilege: Public

UpdateAttributes - This operation updates a list of attributes of the metadata object. This is a multi-scalar version of the UpdateAttribute() operation. Note that it only updates the in-memory part of the object, to make the change persistent, a DsMdCatalog update operation is needed.
Arguments: const GlParameterList& attributes
Return Type: GlStatus
Privilege: Public

~DsMdMetadata - Destructor to deallocate any in-memory data structures for the objects.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsMdMetadata class has associations with the following classes:
DsMdCatalog (Aggregation)

4.3.70 DsNmNMC Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
National Meteorological Center data

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNmNMC class has associations with the following classes:
None

4.3.71 DsNpAncillary Class

Parent Class: DsNsNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class contains ancillary data products which were input in the generation of standard data products. The type attribute inherited from the DsGeESDT is used to indicate which of the ancillary data types an instance of this class is.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNpAncillary class has associations with the following classes:
None

4.3.72 DsNpCalibration Class

Parent Class: DsNsNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class contains instrument and scientific calibration data. The contents and format of this data are not relevant from the DSS perspective. The DSS must receive, store and distribute this data.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNpCalibration class has associations with the following classes:
None

4.3.73 DsNpCorrelative Class

Parent Class: DsNsNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class contains data products that are used as correlative data to evaluate and validate EOS data products.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNpCorrelative class has associations with the following classes:
None

4.3.74 DsNpNonECSDataProduct Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class represents science data that was not produced within ECS. This includes data that is in non-standard formats as well as data that has been produced to recommendations/ specifications provided by ECS to the data producer.

Attributes:

None

Operations:

None

Associations:

The DsNpNonECSDataProduct class has associations with the following classes:
None

4.3.75 DsNpOA Class

Parent Class: DsNsNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This data type contains orbit and attitude data. From the data type perspective of the DSS, the internal format and contents are irrelevant. The DSS services include receive, archive, and distribute for this type.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNpOA class has associations with the following classes:
None

4.3.76 DsNpPlatform Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class represents Satellite Housekeeping Data. The format need not be know to the Data Server.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNpPlatform class has associations with the following classes:
None

4.3.77 DsNpVersion0 Class

Parent Class: DsNsNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class is the base class for all Version 0 data products. At this level, the DSS has no knowledge of the contents and structure of the individual data types. Subclasses derived from this class would be specific Version 0 data types and would require an ESDTDescriptor that defines the type, its services, its metadata, etc.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNpVersion0 class has associations with the following classes:
None

4.3.78 DsNsHistoricalData Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Purpose and Description:
Historical data that the DSS archives and provides access to. This includes instrument historical data and spacecraft historical data. These contain commands sent and indications of success or failure of these commands. From the DSS perspective the contents of these data type are not interpreted but merely archived and made available for distribution.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNsHistoricalData class has associations with the following classes:
None

4.3.79 DsNsMPR Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
The Metadata Problem Report (MPR) is an ESDT that has information submitted by users related to problems with metadata. This information could be supplied to the QA staff via a subscription.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNsMPR class has associations with the following classes:
None

4.3.80 DsNsNonECSDataProduct Class

Parent Class: DsGeScienceData
Public: No Distributed Object: No
Purpose and Description:
This class represents science data that was not produced within ECS. This includes data that is in non-standard formats as well as data that has been produced to recommendations/ specifications provided by ECS to the data producer.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNsNonECSDataProduct class has associations with the following classes:
None

4.3.81 DsNsProdPlans Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Purpose and Description:
This class contains production plans from the planning subsystem. These plans include candidate plans and active plans. From the DSS perspective the format and content of these plans is not relevant.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNsProdPlans class has associations with the following classes:
None

4.3.82 DsNsProductionHistory Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To provide a heritage of an ECS data granule. Product History is an ESDT that denotes the steps that have been taken in the production of a particular ESDT object.

Attributes:

myGranuleUR - The UR for the granule that this production history describes.
Data Type:
Privilege: Private
Default Value:

Operations:

GetGranule - This service provides the ability to obtain the granule that this production history refers to.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsNsProductionHistory class has associations with the following classes:
Class: DsGeScienceData describesgenerationof - Science data is generated using various algorithms on a variety of host machines and with various input data. Production history is used to describe the specific inputs, processing, and environment related to the generation of the associated science data object. Each instance of a science data object has an instance of a production history.

4.3.83 DsNsQAStatistics Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To provide an interface to services for the quality data for a data object. QAStatistics are a type of ESDT that indicates the quality of a particular ESDT object.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsNsQAStatistics class has associations with the following classes:
Class: DsGeScienceData has - An instance of a science data object may or may not have an instance of QA data statistics. An instance of QA data statistics indicates the quality of the associated science data object.

4.3.84 DsNsScienceSoftwareArchivePackage Class

Parent Class: DsGeESDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To provide a single interface to the full content of data and information delivered by data producer standard product Algorithm Integration and Test. The delivered algorithm package (DAP) class is a specialization of the ESDT class. It provides services that are specific to the DAP data type. The DAP is also known as a science software package.

Attributes:

myBinaryList - The set of binary executables that have been compiled for this delivered algorithm package.
Data Type: RWVector<istream>

Privilege: Private
Default Value:

myGranuleList - A list of all the granules that were generated using this science software delivery.
Data Type: RWVector<GlUR>
Privilege: Private
Default Value:

myHostList - The set of hosts that the executable software will run on.
Data Type: RWVector<RWCString>
Privilege: Private
Default Value:

mySourceCode - The source code (ASCII text) of the delivered algorithm package. This will include the programs, header files, and make files.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

AddGranule - This service adds a granule to the list of granules generated using this delivered algorithm package.
Arguments: theUR: GlUR
Return Type: RWBoolean
Privilege: Public

GetBinary - This service provides this algorithm packages executable program which has been compiled for the given host.
Arguments: hostType
Return Type: ostream &
Privilege: Public

GetSource - This service provides the source code from the delivered algorithm package.
Arguments:
Return Type: RWCString
Privilege: Public

HasBinary - This service indicates whether or not the delivered algorithm package has an executable program that runs on the given host type.
Arguments: hostType

Return Type: RWBoolean
Privilege: Public

ListGranules - This service provides a list all of the granules that were generated using this algorithm package.
Arguments:

ListHosts - This service lists all of the host types that this delivered algorithm package has been compiled for.
Arguments: hostType
Return Type: RWVector<RWCString>
Privilege: Public

RemoveGranule - This service removes a granule from the list of granules that were generated using this delivered algorithm package.
Arguments: theUR: GlUR

Associations:

The DsNsScienceSoftwareArchivePackage class has associations with the following classes:
Class: DsGeECSDataProduct isusedtocreate - An instance of a delivered algorithm package (DAP) is used to create zero or more ECS data products. The DAP contains the software, executables, makefiles, etc. The processing subsystem uses the DAP to generate ECS data products. Each ECS data product was created using one DAP.

4.3.85 DsPrRadar Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
The class represents data specifically from the TRMM platform Precipitation Radar (PR) instrument.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsPrRadar class has associations with the following classes:
None

4.3.86 DsSbAction Class

Parent Class: DsSbActionBase
Public: No Distributed Object: No
Purpose and Description:
This defines an activity to be performed on behalf of a server when a previously defined and advertised event occurs. Currently, notifications and requests are valid actions.

Attributes:

All Attributes inherited from parent class

Operations:

DsSbAction - Constructor allowing attributes in the base class to be set.
Arguments: DsSbRequest &, RWBoolean = TRUE, RWCString * = NULL
Return Type: Void
Privilege: Public

DsSbAction - Constructor allowing the attributes in the base class to be set.
Arguments: RWCString &text, DsSbRequest * = NULL
Return Type: Void
Privilege: Public

GetRequest - Returns a subscription request.
Arguments:
Return Type: const DsSbRequest&
Privilege: Public

SetRequest - Sets a request object.
Arguments: const DsSbRequest&

Return Type: void
Privilege: Public

~DsSbAction
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbAction class has associations with the following classes:
DsSbSubscription (Aggregation)

4.3.87 DsSbActionBase Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:

Attributes:

myNotifyFlag - Flag which determines if this action is a notification.
Data Type: RWBoolean
Privilege: Private
Default Value:

myRequestFlag - Flag which determines if this action is a request.
Data Type: RWBoolean
Privilege: Private
Default Value:

myText - Defines text of an action.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

ClearRequest - Provides the action base class the ability to have requests cleared.
Arguments:
Return Type: void
Privilege: Public

DsSbActionBase - Constructs a DsSbActionBase.
Arguments:
Return Type: Void
Privilege: Public

DsSbActionBase - Constructs an Action base class setting the notification flag and text attribute.
Arguments: RWBoolean notify = FALSE, const RWCString *text = 0
Return Type: Void
Privilege: Public

GetNotify - Accesses the notification flag.
Arguments:
Return Type: RWBoolean
Privilege: Public

GetText - Accesses the text attribute.
Arguments:
Return Type: const RWCString &
Privilege: Public

HasRequest - Checks whether the action contains requests.
Arguments:
Return Type: RWBoolean
Privilege: Public

SetNotify - Sets notification flag.
Arguments: RWBoolean
Return Type: void
Privilege: Public

SetText - Sets the text attribute.
Arguments: RWCString &
Return Type: void
Privilege: Public

~DsSbActionBase
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbActionBase class has associations with the following classes:
None

4.3.88 DsSbCallBackTimer Class

Parent Class: DsSbTimer
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To generate calls to another object after a specified interval.

Attributes:

myCallbackFunction - The (member) function that should be called every time the interval occurs.
Data Type: GlCallback
Privilege: Private
Default Value:

Operations:

SetCallback - Used by the managing object to register its callback function.
Arguments:
Return Type: RWBoolean
Privilege: Public

Associations:

The DsSbCallBackTimer class has associations with the following classes:
Class: DsSbSubscriptionInterface alarms

4.3.89 DsSbEvent Class

Parent Class: Not Applicable
Public: No Distributed Object: Yes
Persistent Class: True
Purpose and Description:
Defines events used by the server.

Attributes:

myCategory - This is a category which allows events of the same name to be distinguished from each other.
Data Type: RWCString
Privilege: Private
Default Value:

myDescription - This is the description of the event.
Data Type: RWCString
Privilege: Private
Default Value:

myEventID - This is the event identification number.
Data Type: int
Privilege: Private
Default Value:

myName - This is the name of the event.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

DsSbEvent - Constructs an event.
Arguments:
Return Type: Void
Privilege: Public

Register - Used to register a new event in the subscription server.
Arguments:
Return Type: RWBoolean
Privilege: Public

Trigger - Implements the triggering of an event.
Arguments: GlParameterList&
Return Type: RWBoolean
Privilege: Public
PDL:CALL DsSbEventHandler::DsSbEventHandler to create an event handler for this event.

CALL DsSbEventHandler::EventOccured to indicate event has occured and parameters containing information about the event is passed along.

Returns a success or failure status.

Unregister - Used to remove the registration information about this event from the subscription server.
Arguments:
Return Type: RWBoolean
Privilege: Public

~DsSbEvent
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbEvent class has associations with the following classes:
Class: DsSbEventHandler processedby
DsSbEventTimer (Aggregation)

4.3.90 DsSbEventHandler Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:

Purpose and Description:
Handles incoming asynchronous events as they occur. It activates the corresponding DsSbRegisteredEvent when DsSbEvents are given to it.

Attributes:

None

Operations:

DsSbEventHandler
Arguments:
Return Type: RWBoolean
Privilege: Public

EventOccurred - Indicates that an event has occured and eventually subscriptions against the events are processed.
Arguments: GlParameterList&
Return Type: RWBoolean
Privilege: Public

ReportStatus - Used to return status of execution to calling object.
Arguments:
Return Type: Void
Privilege: Public

~DsSbEventHandler
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbEventHandler class has associations with the following classes:
Class: DsSbRegisteredEvent notifies - The event handler finds and calls the registered event corresponding to an event that just occurred to execute subscriptions.
Class: DsSbEvent processedby

4.3.91 DsSbEventTimer Class

Parent Class: DsSbTimer
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To maintain and generate time events to be submitted to the DsSdEventHanler at regular intervals.

Attributes:

All Attributes inherited from parent class

Operations:

RegisterEvent - Used by the managing object to tell DsSdEventTimer to register its event.
Arguments:
Return Type: RWBoolean
Privilege: Public
PDL:CALL DsSbEvent::DsSbEvent to create a new event and fill in event attributes.

CALL DsSbEvent::Register to advertise an event and record it into the subscription system.

Returns a success or failure status.

Associations:

The DsSbEventTimer class has associations with the following classes:
Class: DsSbSubscriptionInterface managedby - DsSdEventTimers are created and configured by the DsSdSubscriptionInterface to perform regular subscription activities.

4.3.92 DsSbFactory Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:

Attributes:

None

Operations:

DsSbFactory - Constructs a subscription server factory.
Arguments:
Return Type: Void
Privilege: Public

MakeEvent - Used to construct a server-side event to be distributed with a new client-side proxy event.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsSbFactory - Destroys this subscription server factory.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbFactory class has associations with the following classes:
None

4.3.93 DsSbRegisteredEvent Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
Associates subscriptions with subscribable events. It executes all subscriptions that have been registered for an event when that event occurs.

Attributes:

myEventId - Associates this DsSdRegisteredEvent with a DsSdEvent, so that the DsSdEventHandler can find the right DsSdRegisteredEvent when an event is triggered.

Data Type: RWCString
Privilege: Private
Default Value:

Operations:

AddSubscription - Adds a subscription to be executed each time the event associated with this DsSdRegisteredEvent is triggered.
Arguments:
Return Type: RWBoolean
Privilege: Public

CancelSubscription - Used to remove a specific subscription from the list associated with this DsSdRegisteredEvent.
Arguments:
Return Type: RWBoolean
Privilege: Public

CancelSubscriptions - Used to remove all subscriptions from the list of subscriptions associated with this DsSdRegisteredEvent.
Arguments:
Return Type: RWBoolean
Privilege: Public

DsSbRegisteredEvent
Arguments:
Return Type: Void
Privilege: Public

ExecuteSubscriptions
Arguments: GlParameterList
Return Type: RWBoolean
Privilege: Public

GetSubscriptions - Creates a list of all subscriptions associated with this DsSdRegisteredEvent.
Arguments:
Return Type: RWBoolean
Privilege: Public

~DsSbRegisteredEvent
Arguments:

Return Type: Void
Privilege: Public

Associations:

The DsSbRegisteredEvent class has associations with the following classes:

Class: DsSbSubscription activates - Each DsSdRegisteredEvent maintains a list of subscriptions that have been associated with its event, organized by event. The subscription list is managed and subscriptions are executed via this association.

Class: DsSbSubscriptionInterface manages - DsSdSubscriptionInterface maintains a list of all DsSdRegisteredEvents(s) and passes on all subscription requests to the appropriate one.

Class: DsSbEventHandler notifies - The event handler finds and calls the registered event corresponding to an event that just occurred to execute subscriptions.

4.3.94 DsSbSubscription Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
Records all characteristics of a single subscription. It is responsible for executing the requested action when the associated event occurs.

Attributes:

myExpirationDate - Identifies when this subscription will expire and be removed from the system. The value may be "never" (i.e. the subscription is permanent).
Data Type: RWDate
Privilege: Private
Default Value:

myType - Stores the type of this subscription, i.e. whether it is a one-time only or repeating subscription.
Data Type: RWCString
Privilege: Private
Default Value:

myUserId - Identifies the user that submitted the subscription.
Data Type: RWCString

Privilege: Private
Default Value:

Operations:

DsSbSubscription -
Arguments:
Return Type: Void
Privilege: Public

Execute - Implements execution of the server side subscription.
Arguments: GlParameterList
Return Type: RWBoolean
Privilege: Public

NotifyCancel - Used to notify the client that the server side subscription has been cancelled.
Arguments:
Return Type: RWBoolean
Privilege: Public

NotifyExpiration - Used to notify the client that the server side subscription is about to expire.
Arguments:
Return Type: RWBoolean
Privilege: Public

SetExpiration - Sets the expiration of the server side subscription.
Arguments: Date
Return Type: RWBoolean
Privilege: Public

~DsSbSubscription -
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbSubscription class has associations with the following classes:
Class: DsSbRegisteredEvent activates - Each DsSdRegisteredEvent maintains a list of subscriptions that have been associated with its event, organized by event. The

subscription list is managed and subscriptions are executed via this association.
Class: DsSbSubscriptionInterface manages - DsSdSubscriptionInterface maintains a list of all DsSdSubscriptions orgainized by expiration data and user id.

4.3.95 DsSbSubscriptionInterface Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
Provides an interface to subscription services. Other objects can use it to access all administrative functions that are necessary for creating, removing and managing subscriptions. It also handles event registration, and returns a handle to the DsSdEventHandler to registering events.

Attributes:

None

Operations:

AddSubscription - Creates a new DsSbSubscription object for the client and adds it to the appropriate DsSbRegisterEvent (to be used when the correponding event is triggered).
Arguments:
Return Type: RWBoolean
Privilege: Public

CancelSubscription - Notifies the appropriate DsSbRegisteredEvent to remove the subscription from its list (and destroy the DsSdSubscription object).
Arguments:
Return Type: RWBoolean
Privilege: Public

CheckExpirations - Cycles through the entire list of DsSbSubscriptions, and cancels any subscriptions that have expired (with notification to the owner). Also, sends owners notification of subscriptions that will soon expire.
Arguments:
Return Type: RWBoolean
Privilege: Public

DsSbSubscriptionInterface
Arguments:
Return Type: Void
Privilege: Public

ExecuteCommand - Used to cause DsSbSubscriptionInterface to execute a DsSbCommand.
Arguments:
Return Type: RWBooolean
Privilege: Public

GetAllEvents - Creates (and returns) a list of all subscribable events.
Arguments:
Return Type: RWBoolean
Privilege: Public

GetAllSubscriptions - Creates (and returns) a list of all subscriptions that exist for a given user.
Arguments:
Return Type: RWBoolean
Privilege: Public

RegisterEvent - Creates a DsSbRegisteredEvent object for the given event, and returns to the calling object a handle to the DsSbEventHandler to be used when triggering the event.
Arguments:
Return Type: RWBoolean
Privilege: Public

UnregisterEvent - Removes a DsSbRegisteredEvent from the system, and thereby all of the subscriptions associated with it.
Arguments:
Return Type: RWBoolean
Privilege: Public

UpdateSubscription - Modifies any/all attributes of an existing subscription.
Arguments:
Return Type: RWBoolean
Privilege: Public

~DsSbSubscriptionInterface
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbSubscriptionInterface class has associations with the following classes:

Class: DsSbCallBackTimer alarms

Class: DsSbEventTimer managedby - DsSdEventTimers are created and configured by the DsSdSubscriptionInterface to perform regular subscription activities.

Class: DsSbRegisteredEvent manages - DsSdSubscriptionInterface maintains a list of all DsSdRegisteredEvents(s) and passes on all subscription requests to the appropriate one.

Class: DsSbSubscription manages - DsSdSubscriptionInterface maintains a list of all DsSdSubscriptions orgainized by expiration data and user id.

4.3.96 DsSbTimer Class

Parent Class: Not Applicable

Public: No Distributed Object: No

Persistent Class:

Purpose and Description:

To generate time events to be submitted to the subscription system. A timer will monitor the system clock and generate the specified time events.

Attributes:

myInterval - Identifies how much time the timer should wait until generating the next trigger.

Data Type: float

Privilege: Private

Default Value:

Operations:

DsSbTimer

Arguments:

Return Type: Void

Privilege: Public

Reset - Used to tell the timer to start measuring the interval as if it had just been triggered.

Arguments:

Return Type: RWBoolean

Privilege: Public

~DsSbTimer
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSbTimer class has associations with the following classes:
None

4.3.97 DsSd24BitImage Class

Parent Class: DsSdImage
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
A computer science data type used for the representation of 24 bit images.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSd24BitImage class has associations with the following classes:
None

4.3.98 DsSd8BitImage Class

Parent Class: DsSdImage
Public: No Distributed Object: No
Persistent Class:

Purpose and Description:
A computer science data type used the representation of 8 bit images.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSd8BitImage class has associations with the following classes:
None

4.3.99 DsSdCSDT Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
Computer Science Data Type. The CSDT is a data structure used by the data server superclass that provides common services for the different types of CSDTs. The CSDT provides the internal representation of data objects.

Attributes:

myCompressionType
Data Type: typedef
Privilege: Private
Default Value: None

myDate - The date when the CSDT was created.
Data Type: char
Privilege: Private
Default Value: yyyy/mmm/ddd

myFormat - Ths format of the CSDT.
Data Type: typedef

Privilege: Private
Default Value: Raw

myPermissions
Data Type: char
Privilege: Private
Default Value: RW

mySize - The block size of the CSDT
Data Type: integer
Privilege: Private
Default Value: 0

myVersionNo

Operations:

DeleteCSDT
Arguments:

Extract
Arguments:

GetFormat
Arguments:
Return Type: typedef
Privilege: Public
This is an abstract operation

GetLocation
Arguments:

GetPermissions
Arguments:

GetSize - To display the size of the CSDT.
Arguments:

GetVersionNo
Arguments:

GetView
Arguments:

Return Type: Void
Privilege: Public
This is an abstract operation

Insert -
Arguments:

NewCSDT
Arguments:

SetPermissions
Arguments:

UpdateCSDT
Arguments:

Associations:

The DsSdCSDT class has associations with the following classes:
DsGeESDT (Aggregation)

4.3.100DsSdGrid Class

Parent Class: DsSdCSDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
A computer science data type used for the representation of granules in the geolocated data types.

Attributes:

myArrayLabels

myCompression

myDimension

myGeophysicalParameters

myGridType

myName

myResolution

Operations:

ApplyProjection - To apply projection on the granules.
Arguments:
Return Type: Void
Privilege: Public

Compress
Arguments:

ExtractSlice - This service obtains one or more parameters (a slice) over the desired area of interest.
Arguments:

GetOrbitModelName
Arguments:

Subsample - To select representative measurements across rectangle from data granules using a consistant sampling scheme.
Arguments: rectangle
Return Type: Void
Privilege: Public

Subsample - To select representative measurements across polygon from data granules using a consistant sampling scheme.
Arguments: polygon
Return Type: Void
Privilege: Public

Subset - To obtain data from granules based on a given rectangle.
Arguments: rectangle
Return Type: Void
Privilege: Public

Subset - To obtain data from granules based on a given 3-D polygon.
Arguments: polygon

Return Type: Void
Privilege: Public

Subset - To obtain only specified parameters or parameter catagory from granule.
Arguments: parameter
Return Type: Void
Privilege: Public

Subset - To obtain data based on given time(s) or time interval(s).
Arguments: time
Return Type: Void
Privilege: Public

Uncompress
Arguments:

Associations:

The DsSdGrid class has associations with the following classes:
None

4.3.101DsSdImage Class

Parent Class: DsSdCSDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
A computer science data type used for the representation of the 2D raster data type, primarily for the storage of the pixels for image visualization.

Attributes:

myBitDepth

myImageType - The type (bitmap, rastermap etc.,) of the image.
Data Type:
Privilege: Private
Default Value:

myLength - The length of the image.
Data Type:
Privilege: Private
Default Value:

myName

myResolution - The pixel resolution of the image.
Data Type:
Privilege: Private
Default Value:

myWidth - The width of the image.
Data Type:
Privilege: Private
Default Value:

Operations:

Animate
Arguments:

Compress
Arguments:

Overlay - To overlay the image.
Arguments:
Return Type: Void
Privilege: Public

Subsample
Arguments: rectangle

Subset
Arguments: rectangle

Uncompress
Arguments:

Associations:

The DsSdImage class has associations with the following classes:
None

4.3.102DsSdLookUpTable Class

Parent Class: Not Applicable

Attributes:

myColorNames

myName

Operations:

SetColor
Arguments: id

SetContrast
Arguments:

Associations:

The DsSdLookUpTable class has associations with the following classes:
DsSd8BitImage (Aggregation)

4.3.103DsSdPoint Class

Parent Class: DsSdCSDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
A computer science data type used for the representation of the point data.

Attributes:

my size

mylables

mylocation

myname

myresolution

mytime

myvector

Operations:

GetLabels
Arguments:

GetRecordType
Arguments:

GetSize
Arguments:

ParameterSubset
Arguments:

SetLabels
Arguments:

Associations:

The DsSdPoint class has associations with the following classes:
None

4.3.104DsSdRaw Class

Parent Class: DsSdCSDT

Attributes:

mylocation

myname

mysize

Operations:

GetSize
Arguments:

Associations:

The DsSdRaw class has associations with the following classes:
None

4.3.105DsSdSwath Class

Parent Class: DsSdCSDT
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
A computer science data type used for the representation of the swath of the images.

Attributes:

myCompression

myDimensions

myLabels

myLocation

myName

myOrbitRepresentation

myResolution

mySize

mySwathType

myTime

Operations:

Compress
Arguments:

CreateRows
Arguments:

DeleteRows
Arguments:

ExtractRows
Arguments:

GetRecordType
Arguments:

GetSize
Arguments:

Subsample - To select representative measurements across scanline from data granules using a consistant sampling scheme.
Arguments: scanline
Return Type: Void
Privilege: Public

Subsample - To select representative masurements across time from data granules using a consistant sampling scheme.
Arguments: time

Return Type: Void
Privilege: Public

Subsample
Arguments: row

Subset - To obtain data based on given parameters or parameter catagory from garnule.
Arguments: parameter
Return Type: Void
Privilege: Public

Uncompress
Arguments:

Associations:

The DsSdSwath class has associations with the following classes:
None

4.3.06 DsSrClient Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
Acts as an interface class to the CSMS subsystem to use the accounting and user profiling capabilities.

Attributes:

myProfile - A pointer to the profile for this client, obtained from MSS.
Data Type: MSS_UserProifle*
Privilege: Private
Default Value:

mySystemLog - Identifies the log that is used by DsSdClient to log all activity by the object.
Data Type: GlLog
Privilege: Private
Default Value:

Operations:

ConvertToDollars - Used to convert a request to its cost.
Arguments: rq: DsSrRequest &
Return Type: RWBoolean
Privilege: Public

Debit - Used to subtract the given cost from the user's balance.
Arguments: amt: float
Return Type: void
Privilege: Public

DsSrClient - Constructs a client from the given profile.
Arguments: MSS_UserProfile*
Return Type: Void
Privilege: Public

GetBalance - Retrieves the current account balance for this user.
Arguments:
Return Type: float
Privilege: Public

GetProfile - Retrieves the contents of the profile for the user.
Arguments:
Return Type: MSS_UserProfile
Privilege: Public

SufficientFunds - Used to determine if the user has enough money to perform the request.
Arguments: rq: DsSrRequest &
Return Type: RWBoolean
Privilege: Public

~DsSrClient - Destroys this client.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrClient class has associations with the following classes:
Class: DsSrConnection informs

4.3.107DsSrCommand Class

Parent Class: DsSrCommandBase
Public: No Distributed Object: No
Persistent Class: True
Purpose and Description:
An implementation of the DsCommand abstract base class that provides services necessary for data server creation and storage of command objects.

Attributes:

All Attributes inherited from parent class

Operations:

DsSrCommand - Used to construct a command from a command info object. DsSrConnection uses this constructor to rebuild a command that has been submitted from a client.
Arguments: DsSrCommandInfo*
Return Type: Void
Privilege: Public

~DsSrCommand - Used to destroy a command.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrCommand class has associations with the following classes:
DsSrRequest (Aggregation)

4.3.108DsSrCommandBase Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:

Purpose and Description:
This class provides a common interface to the core information of a command in the system. It is inherited by both the client-side and server-side command classes.

Attributes:

myInfo - A pointer to the core command information for this command. (Which is, In OO terms, the implementation for this interface).
Data Type: DsSrCommandInfo*
Privilege:
Default Value:

Operations:

DsSrCommandBase - Used to construct an empty command.
Arguments:
Return Type: Void
Privilege: Public

DsSrCommandBase - Used to construct a command with the given service, parameters, and category.
Arguments: svc: RWCString &, pl: GlParameterList *, DsESrCommandCategory
Return Type: Void
Privilege: Public

GetCategory - Used to retrieve the current category for this command.
Arguments:
Return Type: DsESrRequestCategory
Privilege: Public

GetInfo - Returns a pointer to the underlying data (DsSrRequestInfo) for this request.
Arguments:
Return Type: DsSrCommandInfo*
Privilege: Public

GetParameters - Returns the parameters for this command.
Arguments:
Return Type: const GlParameterList &
Privilege: Public

GetServiceName - Returns the current service name for this command.
Arguments:

Return Type: const RWCString &
Privilege: Public

SetCategory - Used to set the category for the command.
Arguments: DsESrRequestCategory
Return Type: void
Privilege: Public

SetParameters - Used to set the parameters for this command.
Arguments: pl: GlParameterList &
Return Type: void
Privilege: Public

SetServiceName - Used to set the service name for this command.
Arguments: svc: const RWCString &
Return Type: void
Privilege: Public

~DsSrCommandBase - Used to destroy a command.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrCommandBase class has associations with the following classes:
Class: DsSrCommandInfo referencedby

4.3.109DsSrCommandInfo Class

Parent Class: Not Applicable

Attributes:

myCategory - The category of the command indicates to the receiving connection object how the command should be routed.
Data Type: DsESrCommandCategory
Privilege: Protected
Default Value:

myParameterList - Designates the parameters values for the service requested.
Data Type: GlParameterList
Privilege: Protected
Default Value:

myServiceName - Indicates the name of the desired service.
Data Type: RWCString
Privilege: Protected
Default Value:

Operations:

None

Associations:

The DsSrCommandInfo class has associations with the following classes:
Class: DsSrCommandBase referencedby
DsSrRequestInfo (Aggregation)

4.3.110DsSrConnection Class

Parent Class: Not Applicable
Public: No Distributed Object: Yes
Persistent Class: True
Purpose and Description:
Manages all interaction between clients and the data server. Provides all necessary server-side housekeeping activities (logging, accounting, authorizing) as well as executing client requests.

Attributes:

myLog - The log used to record all activity for this connection.
Data Type: GlLog
Privilege: Protected
Default Value:

myLoggingFlag - Indicates whether activity is currently being logged for this connection.
Data Type: RWBoolean

Privilege: Protected
Default Value:

myStatus - The current status of this connection.
Data Type: GlStatus
Privilege: Protected
Default Value:

myTimeOut - The time when this connection will automatically expire unless activity is detected.
Data Type: RWDate
Privilege: Protected
Default Value:

myTimeoutInterval - The amount of time (in seconds) after which this connection will terminate if no activity is detected.
Data Type: unsigned int
Privilege: Protected
Default Value:

Operations:

Authorize
Arguments: DsSrCommand
Return Type: GlStatus
Privilege: Public

DsSrConnection - Constructs this connection for the given user profile. If the connection ID is not 0, restores the previously suspended connection.
Arguments: prof: MSS_UserProfile, DsESrConnectionID = NULL
Return Type: Void
Privilege: Public

Execute - A pure-virtual function that is expected to be implemented by derived classes to execute a request.
Arguments: rq: DsSrRequest &
Return Type: GlStatus
Privilege: Public

GetStatus - Returns the status of this connection.
Arguments:
Return Type: GlStatus
Privilege: Public

GetTimeOut - Returns the time at which this connection will automatically terminate without activity.
Arguments:
Return Type: RWTime
Privilege: Public

Refresh - Used to tell the connection to reload any configuration parameters that it uses, usually so that it notices a change. myTimeoutInterval is a parameter that is loaded from configuration.
Arguments:
Return Type: void
Privilege: Public

SetTimeOut - Used to set the interval when this connection will automatically terminate if no activity is detected.
Arguments: secs: unsigned int
Return Type: void
Privilege: Public

StartLogging - Used to make this connection start logging all activity.
Arguments:
Return Type: void
Privilege: Public

StopLogging - Used to make this connection stop logging all activity.
Arguments:
Return Type: void
Privilege: Public

Submit - Used to submit a request to this connection. The request is evaluated to determine its destination (within the data server), and its commands are passed to the appropriate object.
Arguments: rq: DsClSubmittedRequestID
Return Type: GlStatus
Privilege: Public
PDL:CALL DsSrServer::GetSRVector to obtain a reference to the vector of submitted requests
CALL DsSrSubmittedRequestVector::Find to find the DsClSubmittedRequest for the given ID
IF the DsClSubmittedRequest is found
CALL DsClSubmittedRequest::GetInfo to get a reference to the core request information
Construct a new DsSrRequest with the core request information
CALL DsSrRequestVector::insert to add the new request to the vector of requests
ELSE

Set return code to indicate "Invalid request ID"
END IF

Terminate - Used to terminate this connection.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsSrConnection - Used to destory the connection.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrConnection class has associations with the following classes:
Class: DsSbSubscriptionInterface
Class: DsSrClient informs
Class: DsSrServer registers
Class: DsSrRequest routes
Class: DsSrSubmittedRequestVector searches

4.3.111DsSrQueuedConnection Class

Parent Class: DsSrConnection

Attributes:

All Attributes inherited from parent class

Operations:

CancelRequest
Arguments: DsClSubmittedRequestID
Return Type: GlStatus
Privilege: Public

DsSrQueuedConnection - Constructs this connection for the given user profile, and creates a request vector to queue future requests. If the connection ID is not 0, restores the previously suspended connection.

Arguments: prof: MSS_UserProfile, resume: DsESrConnectionID = 0
Return Type: Void
Privilege: Public

GetNextRequest - Returns the next request to be executed from the request vector. The vector is searched for the first HIGH-priority request, and if none is found then the first NORMAL-priority request is returned. If the vector contains only LOW-priority requests, the first one is returned.
Arguments:
Return Type: DsSrRequest*
Privilege: Public

Submit - Overrides function inherited from DsSrConnection to add new request to the vector of requests.
Arguments: DsClSubmittedRequestID
Return Type: GlStatus
Privilege: Public

Suspend - Used to suspend this connection, saving the current state and returning an ID that can be used to resume the connection.
Arguments: ref: DsESrConnectionID &
Return Type: GlStatus
Privilege: Public

~DsSrQueuedConnection - Destroys this connection.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrQueuedConnection class has associations with the following classes:
None

4.3.112DsSrRequest Class

Parent Class: DsSrRequestBase
Public: No Distributed Object: No
Persistent Class: True
Purpose and Description:
This class presents the interface needed by other server objects to the core request

information. It also acts as a vector of DsSrCommand objects that compose this request, and maintains a connection to the client-side request via a DsClSubmittedRequest.

Attributes:

All Attributes inherited from parent class

Operations:

Cancel - Used to cancel a request. If the request is in the request vector, it is removed from the vector. A TBD policy will be used to terminate a currently executing request.
Arguments:
Return Type: RWBoolean
Privilege: Public

DsSrRequest - This class presents the interface needed by other server objects to the core request information. It also acts as a vector of DsSrCommand objects that compose this request, and maintains a connection to the client-side request via a DsClSubmittedRequest.
Arguments: DsSrRequestInfo*
Return Type: Void
Privilege: Public

GetResults - Returns the current results for this request (from the DsCLSubmittedRequest).
Arguments:
Return Type: GlParameterList &
Privilege: Public

GetStatus - Used to retrieve the current status of this request (from the DsClSubmittedRequest).
Arguments:
Return Type: GlStatus&
Privilege: Public

SetStatus - Used to set the status of the request. This status is also updated in the DsClSubmittedRequest that corresponds to this request.
Arguments: GlStatus *
Return Type: void
Privilege: Public

~DsSrRequest - Used to destory a request.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrRequest class has associations with the following classes:
Class: DsSrConnection routes
DsSrRequestVector (Aggregation)

4.3.113DsSrRequestBase Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
An abstract base class that represents a prioritized set of commands to be executed on the data server. Inherits an instantiation of an RWVector.

Attributes:

myInfo - A pointer to the core request information for this request. (In OO terms, the implementation for this interface).
Data Type: DsSrRequestInfo*
Privilege: Protected
Default Value:

mySubmittedRequest - The DsClSubmittedRequest that is associated with this request. Value is NULL until one is created/found.
Data Type: DsClSubmittedRequest *
Privilege: Protected
Default Value: NULL

Operations:

DsSrRequestBase - Constructs a request with the given priority (LOW, NORMAL, HIGH).
Arguments: DsESrRequestPriority = NORMAL
Return Type: Void
Privilege: Public

GetDomain - Returns the current UR vector of this request.
Arguments:

Return Type: GlURVector &
Privilege: Public

GetInfo - Returns a pointer to the underlying data (DsSrRequestInfo) for this request.
Arguments:
Return Type: DsSrRequestInfo*
Privilege: Protected

GetPriority - Returns the current priority of the request.
Arguments:
Return Type: DsESrRequestPriority
Privilege: Public

GetSRID - Used to obtain the ID of the submitted request associated with this request.
Arguments:
Return Type: DsClSubmittedRequestID
Privilege: Protected

GetStatus - Returns a reference to the current status of the request.
Arguments:
Return Type: const GlStatus &
Privilege: Public

SetDomain - Used to set the UR vector for this request.
Arguments: GlURVector &
Return Type: void
Privilege: Public

SetPriority - Used to set the priority of the request.
Arguments: DsESrRequestPriority
Return Type: void
Privilege: Public

~DsSrRequest - Destroys a request.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrRequestBase class has associations with the following classes:
Class: DsClSubmittedRequest

4.3.114DsSrRequestInfo Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:

Attributes:

myCommands - A list of commands which comprise this request.
Data Type: RWTPtrOrderedVector<DsSrCommandInfo>
Privilege: Protected
Default Value:

myDomain - A list of URs (of ESDTs currently in the working collection) to which this request should be applied. If this list is empty, the request will be applied to all ESDTs in the collection.
Data Type: GlURVector
Privilege: Protected
Default Value:

myPriority - The priority of this request: LOW, NORMAL, HIGH
Data Type: DsESrRequestPriority
Privilege: Protected
Default Value:

Operations:

None

Associations:

The DsSrRequestInfo class has associations with the following classes:
DsSrRequestBase (Aggregation)

4.3.115DsSrRequestVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No

Persistent Class:
Purpose and Description:
Manages multiple DsSrRequests for a DsSrQueuedConnection objects. Provides ordering based on priority of DsSrRequests and order of receipt (FIFO).

Attributes:

None

Operations:

None

Associations:

The DsSrRequestVector class has associations with the following classes:
DsSrQueuedConnection (Aggregation)

4.3.116DsSrServer Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
A single DsSrServer object is instantiated in each data server. Its main job is to register and keep track of connections. It also maintains a vector of current submitted requests.

Attributes:

myConnectionThreshold - The maximum number of connections that can be started by this server.
Data Type: int
Privilege: Private
Default Value:

mySystemLog - Used to log general system activities.
Data Type: GlLog
Privilege: Private
Default Value:

Operations:

Connect - Used to register a connection for this server.
Arguments: DsSrConnection &
Return Type: GlStatus
Privilege: Public

DeleteConnection - Used to terminate and delete an existing connection.
Arguments: conn: DsSrConnection&
Return Type: GlStatus
Privilege: Public

DsSrServer - Used to construct the DsSrServer object using the given configuration.
Arguments: cf: DsSrConfig
Return Type: Void
Privilege: Public

GetConnectionThreshold - Used to retrieve the current connection threshold.
Arguments:
Return Type: int
Privilege: Public

GetSRVector - Used to obtain a reference to the vector of all active submitted requests. The vector can be modified or searched by the caller.
Arguments:
Return Type: DsSrSubmittedRequestVector &
Privilege: Public

ListConnections - Returns a list of all (currently active) connections known by the server.
Arguments:
Return Type: const DsSrConnectionVector
Privilege: Public

ReportStatus - Returns the current status of the server.
Arguments:
Return Type: GlStatus
Privilege: Public

SetConnectionThreshold - Used to set the maximum number of connections that can be active for this server.
Arguments: int
Return Type: void
Privilege: Public

~DsSrServer - Used to destroy the server object.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrServer class has associations with the following classes:
Class: DsSrConnection registers

4.3.117DsSrSession Class

Parent Class: DsSrQueuedConnection
Public: No Distributed Object: No
Persistent Class:
Purpose and Description:
To manage ESDT-oriented interaction between the external client and the data server. It provides an interface for the other objects in the system to communicate with the client and manages a DsSdWorkingCollection on behalf of the client. Adds request queueing and DsSdWorkingCollection handling functionality to DsSdConnection (its super class).

Attributes:

All Attributes inherited from parent class

Operations:

DsSrSession - Constructs this connection for the given user profile, and creates an empty working collection. If the connection ID is not 0, restores the previously suspended connection.
Arguments: MSS_UserProfile, DsESrConnectionID
Return Type: Void
Privilege: Public

Execute - Executes a request (from the request vector) by passing each command from the request to the working collection. Results and status are accumulated from each command, and returned to the client through the request. Each request may be executed in a separate thread, as long as the domains of the executing requests do not intersect. When a request is found with a domain that intersects with an executing request, it is held until the conflicting request finishes execution.

Arguments: DsSrRequest&
Return Type: GlStatus
Privilege: Public
PDL:CALL GetNextRequest to obtain the next highest-priority request from the request vector
DO WHILE the request has at least one command in it
CALL DsSrRequest::removeFirst to get the first command from the request
CALL Authorize to ensure that the execution of the command is authorized
IF authorization succeeds
Create a results GlParameterList to store the results of the command's execution
CALL DsSrCommand::GetCategory to obtain the category of the request
DO CASE (command category)
CASE INVENTORY:
CASE ESDT:
CASE DISTRIBUTION:
CASE WC:
CALL DsSrWorkingCollection::ExecuteCommand to handle the command

CASE CONNECTION:
Parse the command to determine what function is requested
Execute the function
END DO CASE
CALL DsSrRequest::GetResults to obtain a reference to the results list for the request
CALL GlParameterList::insert to insert the command's results into the request results list
CALL DsSrRequest::SetStatus to indicate that the execution of the command is done
ELSE
RETURN an indication that "Command authorization failed"
END IF
END DO WHILE
CALL DsSrRequest::SetStatus to indicate that the execution of the request is done
Delete the request

Submit - Overrides the function inherited from DsSrQueuedConnection to handle submission of new requests to a session. As requests are added to the request vector, the execution thread is notified of the presence of the new request.
Arguments: DsClSubmittedRequestID
Return Type: GlStatus
Privilege: Public

~DsSrSession - Used to destroy this session.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrSession class has associations with the following classes:
None

4.3.118DsSrSubmittedRequestVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This class is instantiated by the DsSrServer object to maintain a vector of all DsClSubmittedRequests objects that currently exist for this data server. This list can be searched to restore a client's state information after a client shutdown and restart.

Attributes:

None

Operations:

DsSrSubmittedRequestVector - Constructs an empty vector.
Arguments:
Return Type: Void
Privilege: Public

Find - Used to find a specific submitted request in the vector. If none is found, return value will be NULL.
Arguments: DsClSubmittedRequestID
Return Type: DsClSubmittedRequest*
Privilege: Public

~DsSrSubmittedRequestVector - Destroys this vector.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSrSubmittedRequestVector class has associations with the following classes:
Class: DsClSubmittedRequest insertsself

Class: DsSrConnection searches
DsSrServer (Aggregation)

4.3.119DsSrWorkingCollection Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Persistent Class: True
Purpose and Description:
To provide an interface to the current set of ESDT objects. The working collection provides the ability to iterate over this set to perform the requested service. This object facilitates use of a results set as the input domain for further service requests.

Attributes:

myESDTs
Data Type: DsSrESDTWrapperVector
Privilege: Private
Default Value:

myStatus - The current status of this collection.
Data Type: GlStatus
Privilege: Private
Default Value:

Operations:

DsSrWorkingCollection - Constructs an empty working collection.
Arguments:
Return Type: Void
Privilege: Public

ExecuteCommand - Executes the given command over the entire collection, returning results (in the parameter list) and status.
Arguments: DsSrCommand, res: GlParameterList &, GlStatus &
Return Type: Void
Privilege: Public
PDL:CALL DsSrCommand::GetCategory to obtain the category of the command
DO CASE (command category)
CASE WC:

Parse the command to determine what function is requested
Execute the function

CASE ESDT:
DO WHILE (not every ESDT in the collection has executed the command)
Create a GlParameterList to hold the results of the execution for this ESDT
CALL DsGeESDTWrapper::ExecuteCommand to make the ESDT execute the command
CALL GlParamterList::insert to add the ESDT's results to the command results
END WHILE

CASE DISTRIBUTION:
Create a DsDdDistributionList object
Add the data file names for all ESDTs in the collection to the distribution list
CALL DsDdDistributionList::Distribute

CASE INVENTORY:
Create a DsMdCatalog object
CALL DsMdCatalog::Search to search the inventory for ESDTs matchine the criteria
DO WHILE (there are matches in the list returned from Search)
Create a DsGeESDTWrapper object for the corresponding match
END DO WHILE
END DO CASE

ExecuteCommand - Executes the given command over the domain given by the GlURVector, returning results (in the parameter list) and status. The domain must specify URs of ESDTs that are currently contained in the collection.
Arguments: DsSrCommand, GlURVector, res: GlParameterList &, GlStatus &
Return Type: Void
Privilege: Public

GetStatus - Returns the current status of the collection.
Arguments:
Return Type: GlStatus
Privilege: Public

Reset - Used to discard all ESDTs currently held in the collection and return it to the "empty" state.
Arguments:
Return Type: void
Privilege: Public

~DsSrWorkingCollection - Destroys this working collection.
Arguments:

Return Type: Void
Privilege: Public

Associations:

The DsSrWorkingCollection class has associations with the following classes:
DsSrSession (Aggregation)

4.3.120DsSsSSMI Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
The class represents science data generated from the Sea Surface Microwave Imaging instrument (SSMI), this data is primarily used as ancilliary data for the generation of certain TSDIS products.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSsSSMI class has associations with the following classes:
None

4.3.121DsTmTMI Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
This class represents all data derived directly from the TRMM platform Microwave Imager

(TMI) instrument.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsTmTMI class has associations with the following classes:
None

4.3.122DsViVIRS Class

Parent Class: DsNpNonECSDataProduct
Public: No Distributed Object: No
Purpose and Description:
Represents all science data products derived directly from the TRMM platform Visible and Infrared Scanner (VIRS).

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsViVIRS class has associations with the following classes:
None

4.3.123EosHdf24BitImage Class

Parent Class: DsSd24BitImage

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The EosHdf24BitImage class has associations with the following classes:
None

4.3.124EosHdf8BitImage Class

Parent Class: DsSd8BitImage

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The EosHdf8BitImage class has associations with the following classes:
None

4.3.125EosHdfGrid Class

Parent Class: DsSdGrid

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The EosHdfGrid class has associations with the following classes:
None

4.3.126EosHdfLUT Class

Parent Class: DsSdLookUpTable

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The EosHdfLUT class has associations with the following classes:
None

4.3.127EosHdfPoint Class

Parent Class: DsSdPoint

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The EosHdfPoint class has associations with the following classes:
None

4.3.128EosHdfSwath Class

Parent Class: DsSdSwath

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The EosHdfSwath class has associations with the following classes:
None

4.3.129GlBinaryP Class

Parent Class: GlParameter
Public: No Distributed Object: No
Purpose and Description:
This class represents parameters that contain a sequence of binary bytes. It records the binary data and its length.

Attributes:

myBinaryBase - This is a pointer to the actual bytes which comprise this binary parameter.
Data Type: char*
Privilege: Private
Default Value:

myLength - This is the length (in bytes) of the data value of this binary parameter.
Data Type: unsigned long
Privilege: Private
Default Value:

Operations:

GlBinaryP - Constructs an empty binary parameter.
Arguments:
Return Type: Void
Privilege: Public

GlBinaryP - Constructs a binary parameter using the given buffer, length, and name.
Arguments: buffer: char*, length: unsigned long, name: char* = NULL
Return Type: Void
Privilege: Public

Set - Used to set the value of this binary parameter to the given buffer and length.
Arguments: buffer: char*, length: unsigned long
Return Type: void
Privilege: Public

Textify - Used to produce a human-readable form of this parameter in the given string.
Arguments: RWCString&
Return Type: void
Privilege: Public

length - Used to retrieve the length (in bytes) of the binary data that is the value of this parameter.
Arguments:
Return Type: unsigned long
Privilege: Public

value - Used to retrieve the buffer that holds the binary data for this paraemter.
Arguments:
Return Type: char*
Privilege: Public

~GlBinaryP - Used to destroy this binary parameter.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The GlBinaryP class has associations with the following classes:
None

4.3.130GlDateP Class

Parent Class: GlParameter
Public: No Distributed Object: No
Purpose and Description:
This class of parameters holds a single RWDate object.

Attributes:

myDateBase - This is the current value of this date parameter.
Data Type: RWDate
Privilege: Private
Default Value:

Operations:

GlDateP - Constructs a data parameter with a default value and no name.
Arguments:

Return Type: Void
Privilege: Public

GlDateP - Constructs a date parameter with the given name and date.
Arguments: RWDate, name: char* = NULL
Return Type: Void
Privilege: Public

Textify - Used to produce a human-readable form of this parameter in the given string.
Arguments: RWCString&
Return Type: void
Privilege: Public

operator= - Used to set the value of this date parameter.
Arguments: val: RWDate
Return Type: GlDateP&
Privilege: Public

value - Used to retrieve the current value of this date parameter.
Arguments:
Return Type: RWDate
Privilege: Public

~GlDateP - Used to destroy this date parameter.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The GlDateP class has associations with the following classes:
None

4.3.131GlDoubleP Class

Parent Class: GlParameter
Public: No Distributed Object: No
Purpose and Description:
This class of parameters holds a single double value.

Attributes:

myDoubleBase - This is the value of this double parameter.
Data Type: double
Privilege: Private
Default Value:

Operations:

GlDoubleP - Constructs a double parameter with a default value (0) and no name.
Arguments:
Return Type: Void
Privilege: Public

GlDoubleP - Creates a double parameter with the given value and name.
Arguments: double, name: char* = NULL
Return Type: Void
Privilege: Public

Textify - Used to produce a human-readable form of this parameter in the given string.
Arguments: RWCString&
Return Type: void
Privilege: Public

operator= - Used to set a new value for this double parameter.
Arguments: val: double
Return Type: GlDoubleP&
Privilege: Public

value - Used to retrieve the current value of this double parameter.
Arguments:
Return Type: double
Privilege: Public

~GlDoubleP - Used to destroy this double parameter.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The GlDoubleP class has associations with the following classes:
None

4.3.132GlLongP Class

Parent Class: GlParameter
Public: No Distributed Object: No
Purpose and Description:
This class of parameters holds a single long value.

Attributes:

myLongBase - Holds the value for this parameter.
Data Type: long
Privilege: Private
Default Value:

Operations:

GlLongP - Constructs an empty, unnamed long parameter.
Arguments:
Return Type: Void
Privilege: Public

GlLongP - Constructs a long parameter with the given value and name.
Arguments: long, name: char * = NULL
Return Type: Void
Privilege: Public

Textify - Returns a human-readable form of this parameter.
Arguments: RWCString &
Return Type: void
Privilege: Public

operator = - Used to assign a value to a long parameter.
Arguments: val: long
Return Type: GlLongP&
Privilege: Public

value - Returns the current value for this parameter.
Arguments:
Return Type: long
Privilege: Public

~GLongP - Used to destroy a long parameter.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The GlLongP class has associations with the following classes:
None

4.3.133GlParameter Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This is an abstract base class that represents a single parameter that can be passed to many ECS objects. A parameter has a name and an optional description, as well as a value which depends upon its type. Parameters are usually collected together (GlParameterList), and used to dynamically specify values for service calls, result lists, etc. There are several types derived from GlParameter, which implement the value() member function to return an appropriately typed value.

Attributes:

myDescription - The description of this parameter (optional).
Data Type: RWCString
Privilege: Protected
Default Value:

myName - The name of this parameter.
Data Type: RWCString
Privilege: Protected
Default Value:

Operations:

Flatten - Saves an image of this parameter to a binary stream.
Arguments: RWvostream &
Return Type: GlStatus
Privilege: Public
This is an abstract operation

GetDescription - Returns the current name of this parameter as a string.
Arguments:
Return Type: const RWCString &
Privilege: Public

GetName - Returns the name of this parameter as a string.
Arguments:
Return Type: const RWCString &
Privilege: Public

GlParameter - Constructs a parameter with the given name.
Arguments: name: char * = NULL
Return Type: Void
Privilege: Public

Restore - Constructs a parameter from an image stored in a binary stream.
Arguments: RWvistream &
Return Type: GlStatus
Privilege: Public
This is an abstract operation

SetDescription - Sets the name of this parameter.
Arguments: RWCString &
Return Type: void
Privilege: Public

SetName - Sets the name of this parameter.
Arguments: RWCString &
Return Type: void
Privilege: Public

Textify - Returns a human-readable form of the current value of this parameter.
Arguments: RWCString &
Return Type: void

Privilege: Public
This is an abstract operation

saveOn
Arguments: ostream &
Return Type: void
Privilege: Public

~GlParameter - Destroys the parameter.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The GlParameter class has associations with the following classes:
GlParameterList (Aggregation)

4.3.134GlParameterList Class

Parent Class: GlParameter
Public: No Distributed Object: No
Purpose and Description:
This class represents a collection of parameters, and is itself derived from GlParameter. Therefore, GlParameterLists can be embedded in themselves to any depth.

Attributes:

myParms - The actual list of parameters that comprise this GlParameterList.
Data Type: RWOrdered
Privilege: Private
Default Value:

Operations:

FindParameter - Used to search for a parameter in the list with the given name.
Arguments: name: const RWCString &
Return Type: GlParameter *
Privilege: Public

Flatten - Used to save a machine-independent binary image of a parameter list in the given file.
Arguments: filename: RWCString &
Return Type: GlStatus
Privilege: Public

Flatten - Used to save a machine-independent binary image of the parameter list into the given memory buffer.
Arguments: buffer: void *, length: unsigned int
Return Type: GlStatus
Privilege: Public

Flatten - Used to save a machine-independent binary image of a parameter list on the given stream.
Arguments: RWvostream &
Return Type: GlStatus
Privilege: Public

GlParameterList - Constructs an empty, unnamed parameter list.
Arguments:
Return Type: Void
Privilege: Public

GlParameterList - Constructs a parameter list with the given parameter and name.
Arguments: GlParameter *, name: char * = 0
Return Type: Void
Privilege: Public

GlParameterList - Constructs an empty parameter list with the given name.
Arguments: name: char *
Return Type: Void
Privilege: Public

GlParameterList - Constructs a parameter list from a binary stream image.
Arguments: RWvistream &
Return Type: Void
Privilege: Public

Restore - Used to build a parameter list from a machine-independent binary image from the given file.
Arguments: filename: RWCString &
Return Type: GlStatus
Privilege: Public

Restore - Used to build a parameter list from a machine-independent binary image from the given stream.
Arguments: RWvistream&
Return Type: GlStatus
Privilege: Public

Restore - Used to build a parameter list from a machine-independent binary image from the given memory buffer.
Arguments: buffer: void*, length: unsigned int
Return Type: Void
Privilege: Public

~GlParameterList - Used to destroy a paramter list.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The GlParameterList class has associations with the following classes:
None

4.3.135GlStringP Class

Parent Class: GlParameter
Public: No Distributed Object: No
Purpose and Description:
This class of parameters holds a single RWCString object.

Attributes:

myStringBase - This is the value of this string parameter.
Data Type: RWCString
Privilege: Private
Default Value:

Operations:

GlStringP - Constructs an empty, unnamed string parameter.
Arguments:
Return Type: Void
Privilege: Public

GlStringP - Constructs a string parameter with the given value and name.
Arguments: RWCString, name: char* = NULL
Return Type: Void
Privilege: Public

Textify - Used to produce a human-readable form of this parameter in the given string.
Arguments: RWCString&
Return Type: void
Privilege: Public

operator= - Used to assign a new value to this string parameter.
Arguments: val: RWCString
Return Type: GlStringP&
Privilege: Public

value - Used to retrieve a reference to the current value of this string parameter.
Arguments:
Return Type: const RWCString&
Privilege: Public

~GlStringP - Destroys this string parameter.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The GlStringP class has associations with the following classes:
None

4.3.136GlTimeP Class

Parent Class: GlParameter
Public: No Distributed Object: No
Purpose and Description:
This class represents parameters with an RWTime value.

Attributes:

myTimeBase - This is the value of this time parameter.
Data Type: RWTIme
Privilege: Private
Default Value:

Operations:

GlTimeP - Used to construct a time parameter with a default value and no name.
Arguments:
Return Type: Void
Privilege: Public

GlTimeP - Used to construct a time parameter with the given value and name.
Arguments: RWTime, name: char* = NULL
Return Type: Void
Privilege: Public

Textify - Used to produce a human-readable form of this parameter in the given string.
Arguments: RWCString&
Return Type: void
Privilege: Public

operator= - Used to set a new value for this time parameter.
Arguments: val: RWTime
Return Type: GlTimeP&
Privilege: Public

value - Used to retrieve the current value of this time parameter.
Arguments:
Return Type: const RWTime&
Privilege: Public

~GlTimeP - Used to destroy this time parameter.
Arguments:

Return Type: Void
Privilege: Public

Associations:

The GlTimeP class has associations with the following classes:
None

4.3.137MSSLog Class

Parent Class: DsAdLog

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The MSSLog class has associations with the following classes:
None

4.3.138RWTPtrOrderedVector Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description:
This is a COTS class from the Rogue Wave library that can be used to implement ordered lists of objects. It is a template that can be instantiated over any class that implements the == operator.

Attributes:

None

Operations:

None

Associations:

The RWTPtrOrderedVector class has associations with the following classes:
None

4.4 CSCI Dynamic Model

This section describes several scenarios for the Science Data Server CSCI. A scenario delineates the steps that occur in a particular execution through the system. The scenarios address the following topics:

- Acquiring an ESDT
- Asynchronous Status Updates
- Auto-cancel a Subscription
- Canceling a Subscription
- Catalog Deleting a Metadata Entry
- Catalog Insert Collection Metada
- Catalog Insertion of Metadata
- Catalog Search
- Catalog Updating Metadata
- Changing a Request's Priority
- Client Browsing
- Client Connecting to a Data Server
- Client Request Submission
- Client Searching
- Deleting a Queued Request
- Ending Session No Active Requests
- Fulfilling a One-time Subscription
- Fulfilling Open Ended Subscription
- Inserting Composite ESDT
- Inserting a New ESDT

- Inserting Single ESDT
- Instantiating an ESDT
- Operator Viewing Queued Requests
- Registering a Subscribable Event
- Returning a List of Subscriptions
- Server handling a Browse Request
- Server handling of a Search Request
- Server Request Handling
- Startup of a Science Data Server
- Submitting a Subscription
- Unregistering a Subscribable Event
- Update Server Configuration
- Updating a Subscription
- Validating Metadata

Figures 4.4-1 through 4.4-34 are event trace diagrams from OMT that define the class interactions used to support these scenarios.

4.4.1 SDSRV_Acquiring_an_ESDT

Summary

This scenario begins at the point where the server session receives an Acquire command request from the client (refer to scenario: SDRV_Server_Request_Handling). The session determines that the already established working collection (containing one granule) is required to service the command, the working collection in turn passes the command to the granule. The granule now determines the type of service it is required to perform and, if available, calls that service - “Externalize” is the scenario. Two main operations are then performed. First the granule meta-data is extracted from the database and written out to file, and second the granule science data files are copied from the archive. The location and names of these files is returned to the working collection. Following completion of all ESDT “Externalize” services (in this scenario there is only one) the working collection then requests that the files be prepared for distribution.

Assumptions

- A work area has already been assigned.
- A connection has already been established to the data server.
- A search has been performed and a collection already contains one granule.
- An Acquire command request has been generated and submitted by the client to the data server.
- Only one granule is being requested.

- The Acquire command request has been handled be the Session and passed onto the Working collection (see ET SDSRV_Server_Request_Handling).
- The data granule files are passed onto distribution for eventual staging/despatch to the user.

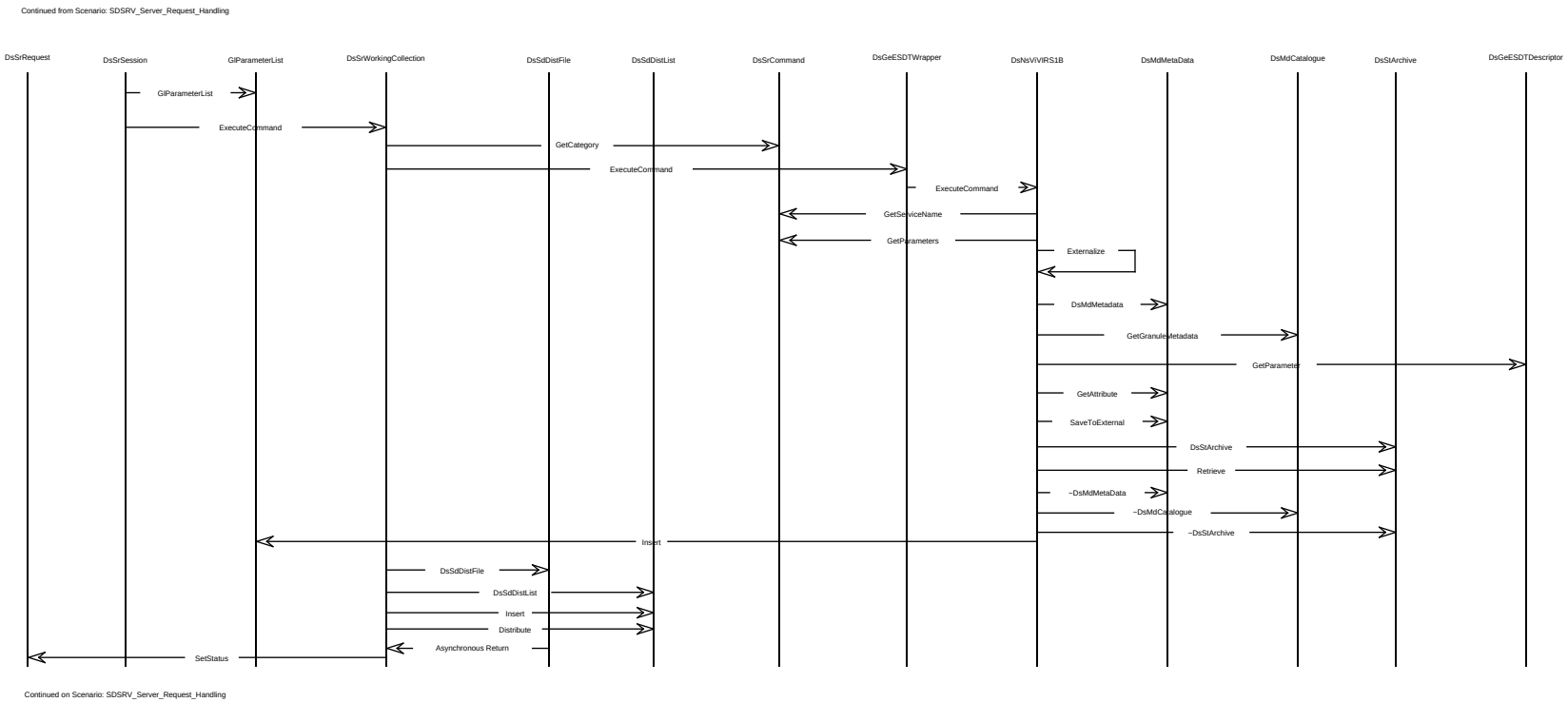


Figure 4.4-1. SDSRV_Acquiring_an_ESDT Dynamic Model

4.4.2 SDSRV_Asynchronous_Status_Updates

Summary

This scenario shows a common mechanism used in client interaction with the data server: the asynchronous reporting of status updates. The basic mechanism is a callback function that is established by the client software that is called every time the status of a the request changes. Thus the client knows immediately of progress statuses (10%, 20%, etc.), error statuses, and when the request finishes (status = DONE). On the diagram, the split between the client and the dataserver (i.e. the network) occurs between the two classes labeled DsClSubmittedRequest. DsClSubmittedRequest is a distributed object whose client-side (the "proxy") forwards function calls to the server-side (the "real" object). Once the client creates a request and establishes a callback, the proxy calls a function on the "real" DsClSubmittedRequest that will not return until a status change occurs in the server ("WaitStatus"). This function runs in its own thread of control so that the request may be canceled at any time. If all goes well, eventually the client receives the DONE status and can obtain the results from the DsClRequest. Notice that the client never deals directly with the DsClSubmittedRequest, only with the DsClRequest which forwards calls to the DsClSubmittedRequest.

Assumptions

- all steps execute successfully
- filling the request with command(s) is left to other scenarios
- submitting the request to a collector is left to other scenarios

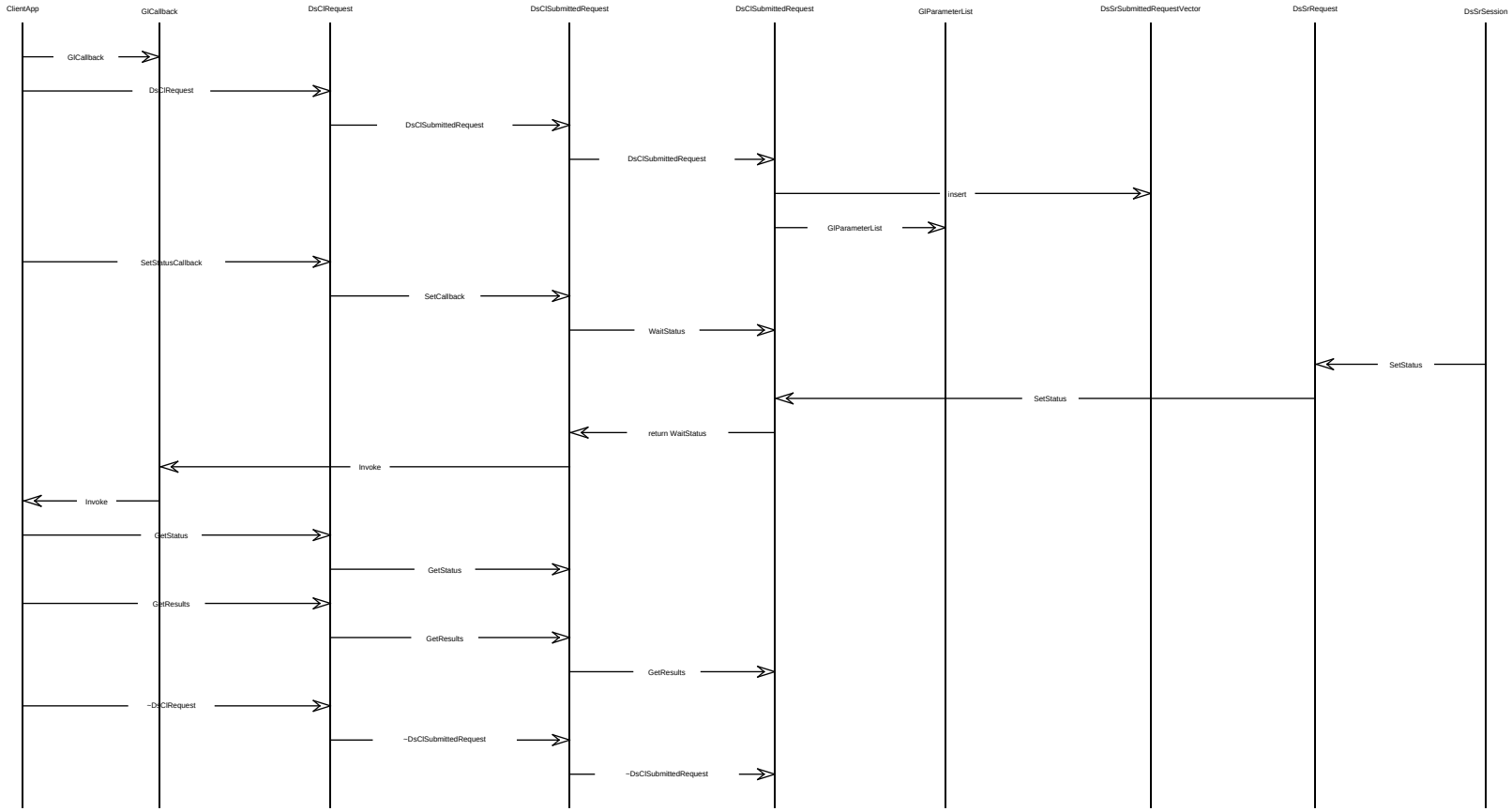


Figure 4.4-2. SDSRV_Asynchronous_Status_Updates Dynamic Model

4.4.3 SDSRV_Auto-cancel_A_Subscription

Summary

This scenario shows the steps that are executed when a subscription expires. Note that the user is notified in advance of a subscription expiration, and it only expires if it's not renewed. The diagram begins with the daily callback timer firing, which calls CheckExpirations on the DsSbSubscriptionInterface object. The DsSbSubscriptionInterface searches the list of DsSbSubscription objects for any that have expired, and when it finds one it calls NotifyCancel on that object. This generates a notification that is sent to the user (either direct or email) informing him of the expiration of the subscription. Finally, the subscription is canceled and removed from the server.

Assumptions

- there is one subscription that has expired today
- the subscription interface callback timer is about to fire
- all steps execute successfully

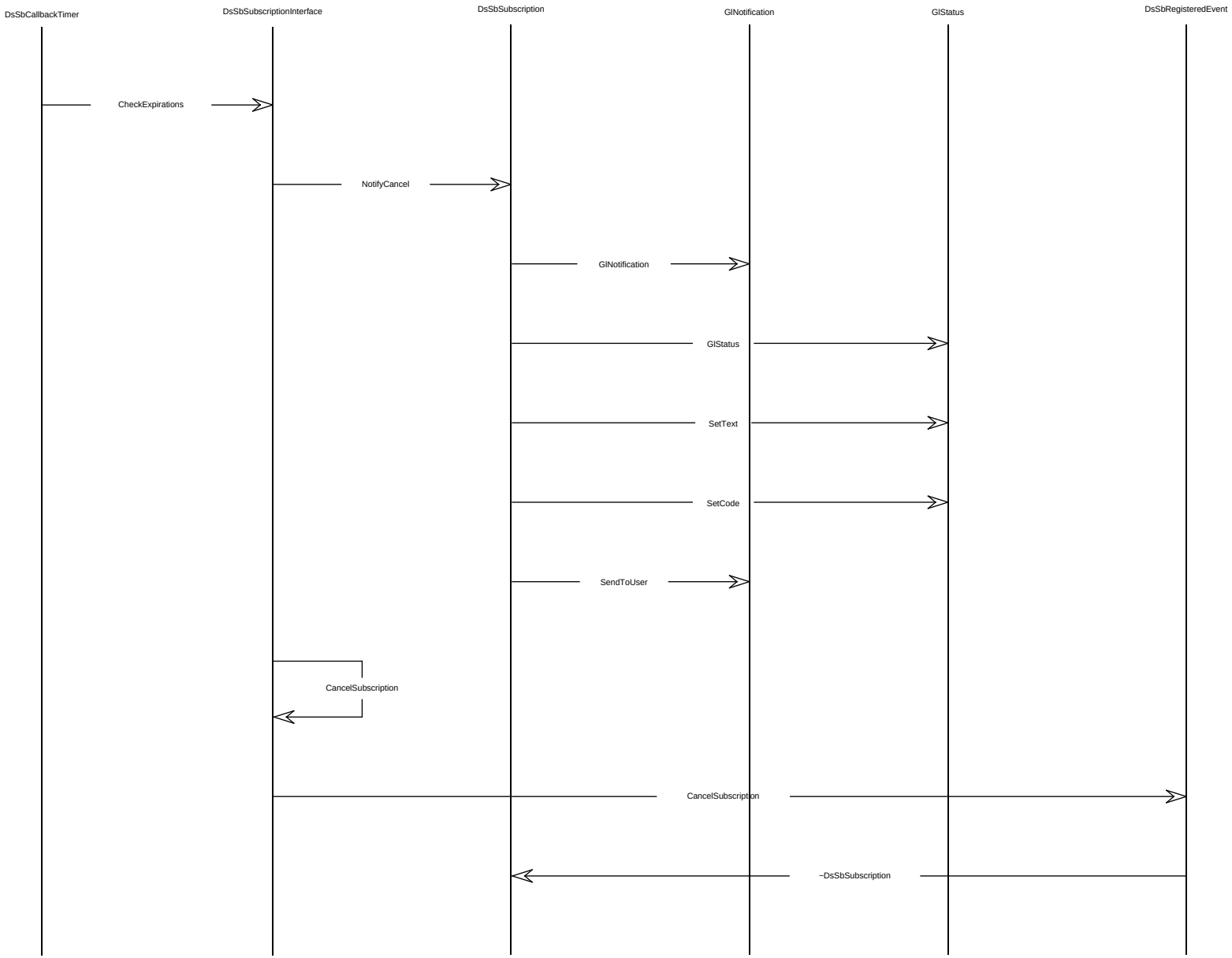


Figure 4.4-3. SDSRV_Auto-Cancel_A_Subscription Dynamic Model

4.4.4 SDSRV_Canceling_a_Subscription

Summary

The scenario shows how the client cancels a subscription. The scenario shows steps used to withdraw a subscription from the client side and then reflected on the server side. This is achieved through the distributed object, DsClSubscriptionCollector. The object, DsClSubscriptionCollector, has already been constructed, and the client implements the withdrawal operation of the client side subscription. All objects on the server side cancel the particular subscription.

Assumptions

- user has logged on and has proper permissions.
- DsClSubscriptionCollector has already been created.

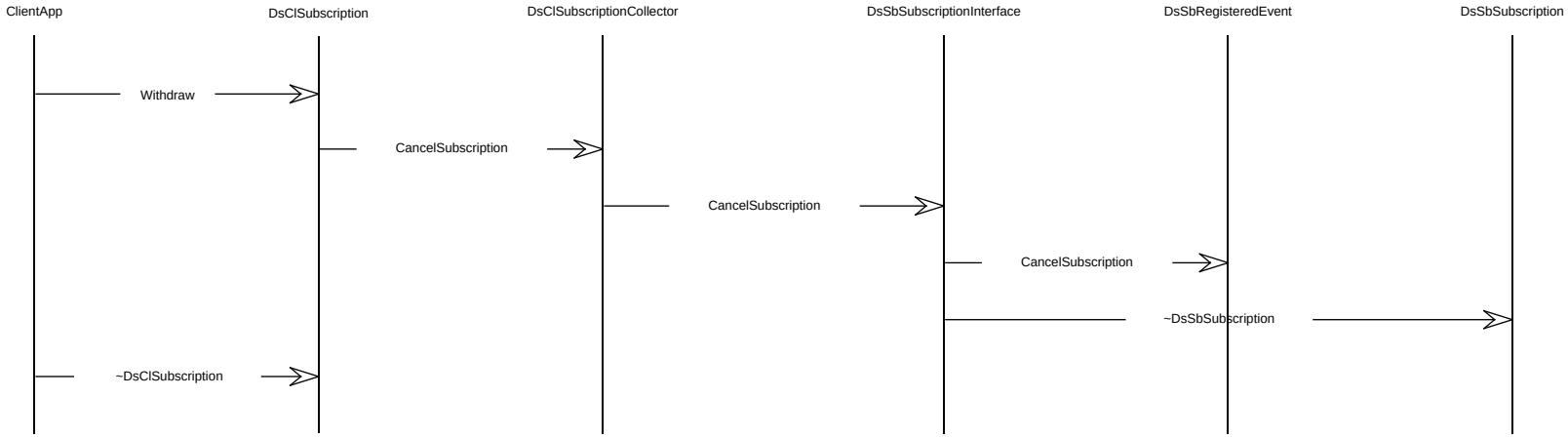


Figure 4.4-4. SDSRV_Canceling_a_Subscription Dynamic Model

4.4.5 SDSRV_Catalog_Deleting_a_Metadata_Entry

Summary

This scenario shows the sequence of events to delete a DsMdMetadata object from the database. The granule identifier is passed to the DeleteGranuleMetadata operation of the DsMdCatalog class. The DsMdCatalog class is responsible for formulating a database delete command. The execution of the command is carried out in the DsDbInterface class. The logic for determining the tables involved in the deletion and the order in which to delete is within the Sybase stored procedures.

Assumptions

- The DsMdMetadata object should already exist in the database before the deletion. The initiator passes in a unique granule identifier. It is also assumed that an instance of the DsMdCatalog is available for processing operations.
- The deletion is persistent. Any subsequent retrieval of the metadata object (either by the same user or other user) will fail.

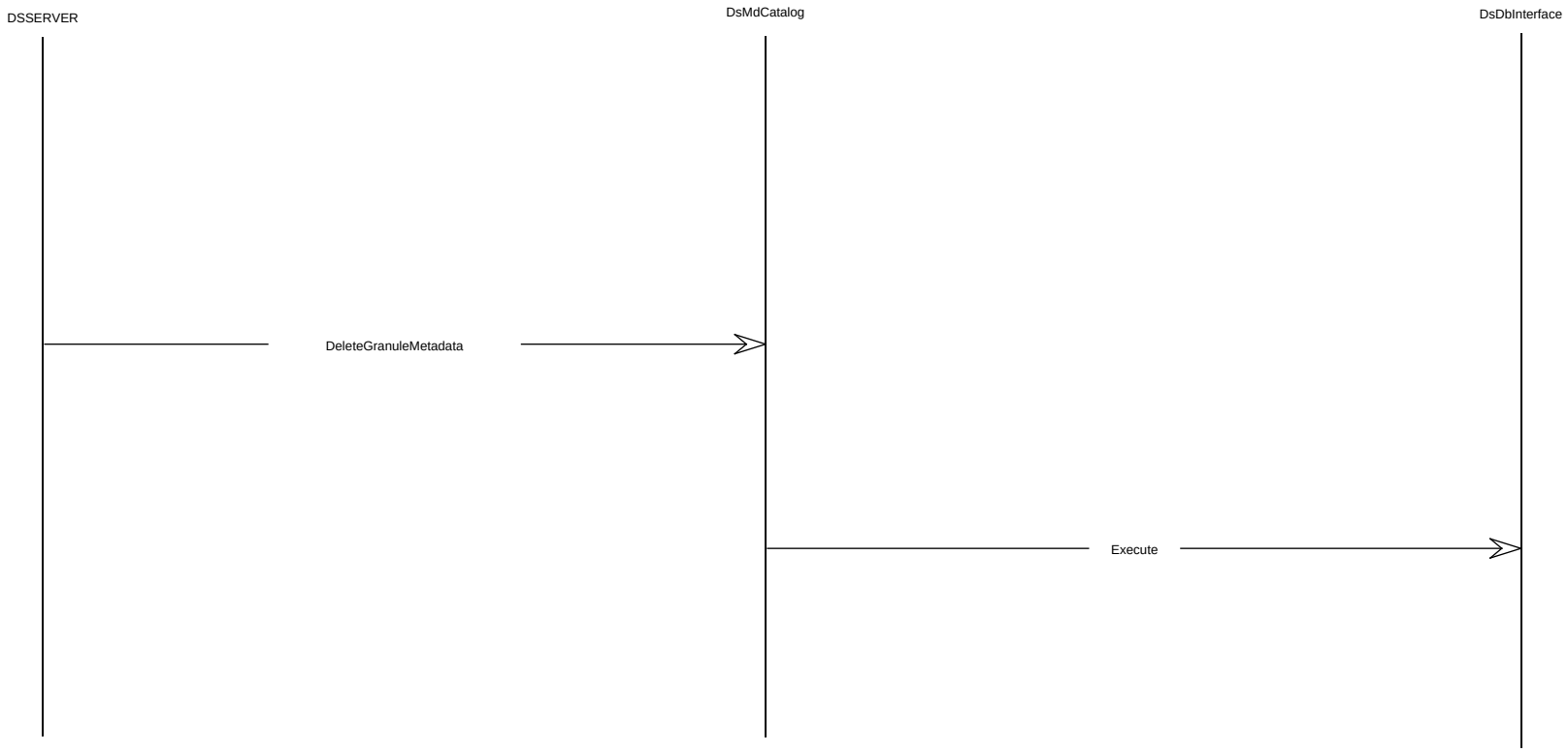


Figure 4.4-5. SDSRV_Catalog_Deleting_a_Metadata_Entry Dynamic Model

4.4.6 SDSRV_Catalog_Insert_Collection_Metadata

Summary

This scenario shows the insertion of collection level metadata as a result of adding a new type to the SDSRV. The DsMdCatalog class supports this operation through the InsertCollectionMetadata method. This method updates the DsDbGranuleToDbVector with the name of the database storing granules for the collection and uses the DsDbInterface class to invoke Sybase stored procedures to insert the data.

Assumptions

- This scenario assumes the initiator must first create a DsMdCatalog which establishes a connection to the database. The scenario also assumes the existence of a PVL string buffer containing all the metadata for the collection to insert and that all attributes are valid for a collection.
- The DsMdCatalog is not deleted and is available for other operations.

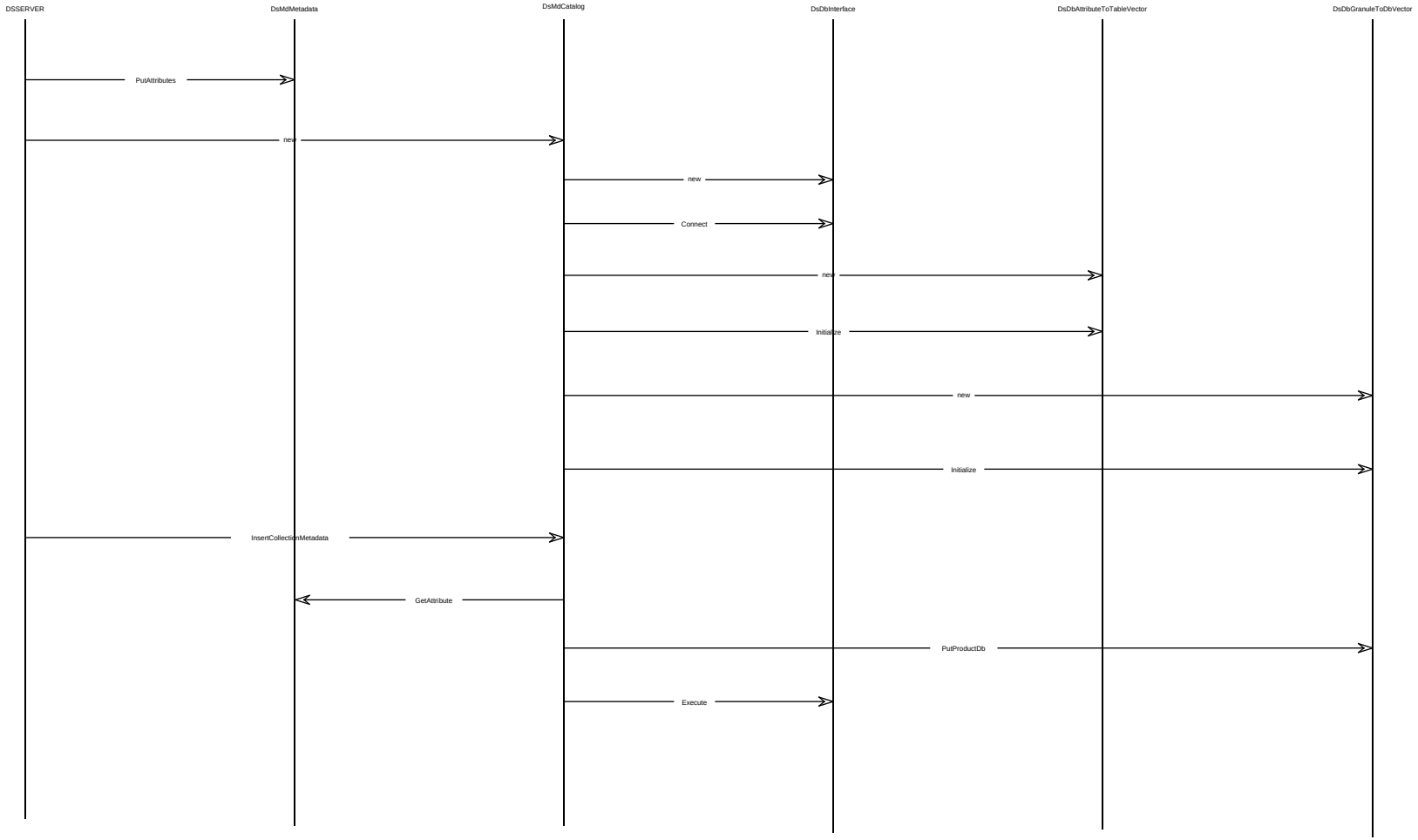


Figure 4.4-6. SDSRV_Catalog_Insert_Collection_Metadata Dynamic Model

4.4.7 SDSRV_Catalog_Insertion_of_Metadata

Summary

This scenario shows how metadata, stored in PVL format, is parsed and inserted into the SDSRV databases. The PVL is captured using the DsMdMetadata LoadFromExternal method. The instance of the DsMdMetadata class is passed to the DsMdCatalog class where attributes are mapped to databases, tables, and column names and inserted into the SDSRV database through the InsertGranuleMetadata method. In addition, the scenario shows the use of the DsDbWrapper classes (DsDbInterface, DsDbGranuleToTableVector, and DsDbAttributeToTableVector) for establishing a connection to Sybase, executing Sybase commands, and mapping granules to the appropriate database.

Assumptions

This scenario assumes the initiator must first create a DsMdCatalog which establishes a connection to the database. The scenario also assumes the existence of a PVL string buffer containing all the metadata for the granule and that all PVL attributes are valid for the product type and contain valid values. The DsMdCatalog is not deleted and is available for other operations.

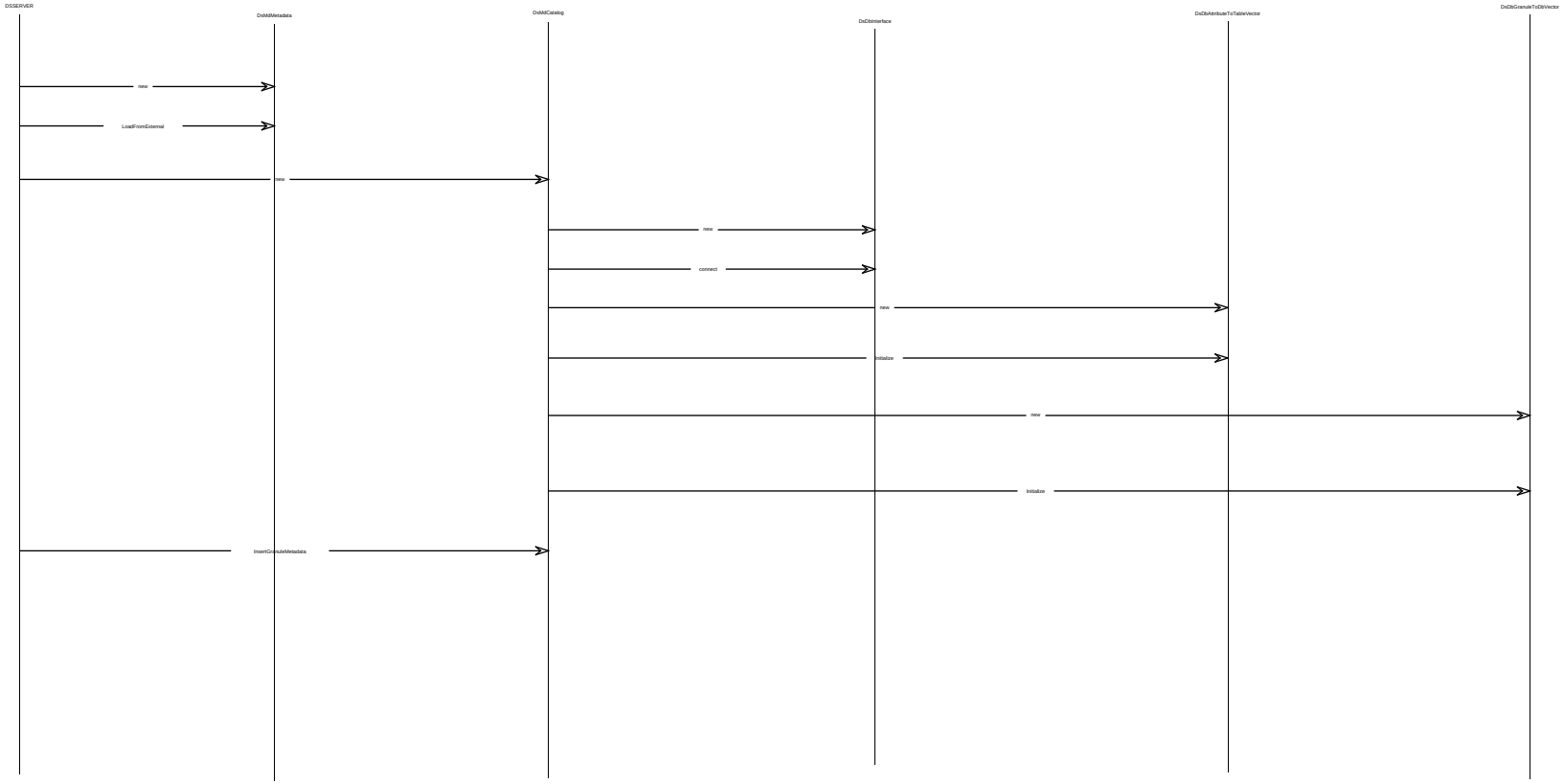


Figure 4.4-7. SDSRV_Catalog_Insertion_of_Metadata Dynamic Model

4.4.8 SDSRV_Catalog_Search

Summary

This scenario shows how a database query, expressed in PVL and stored in a GlParameterList, is executed. It shows how the DsMdCatalog class uses the DsDbWrapper classes (DsDbInterface, DsDbGranuleToTableVector, and DsDbAttributeToTableVector) for establishing a connection to Sybase, executing Sybase commands, determining which databases need to be queried. In addition, these classes map attributes names defined by the DsDeDiscriptor to table and column names in the SDSRV databases. Finally it shows how the metadata attributes are loaded into an instance of the DsMdMetadata class (one per result granule) and packaged in a RWTPtrOrderedVector instance for return to the initiator.

Assumptions

- This scenario assumes the initiator must first create a DsMdCatalog which establishes a connection to the database. The scenario also assumes a GlParameterList is passed in and that the names of the attributes in the list use valid ECS database names.
- The DsMdCatalog is not deleted and is available for other operations.

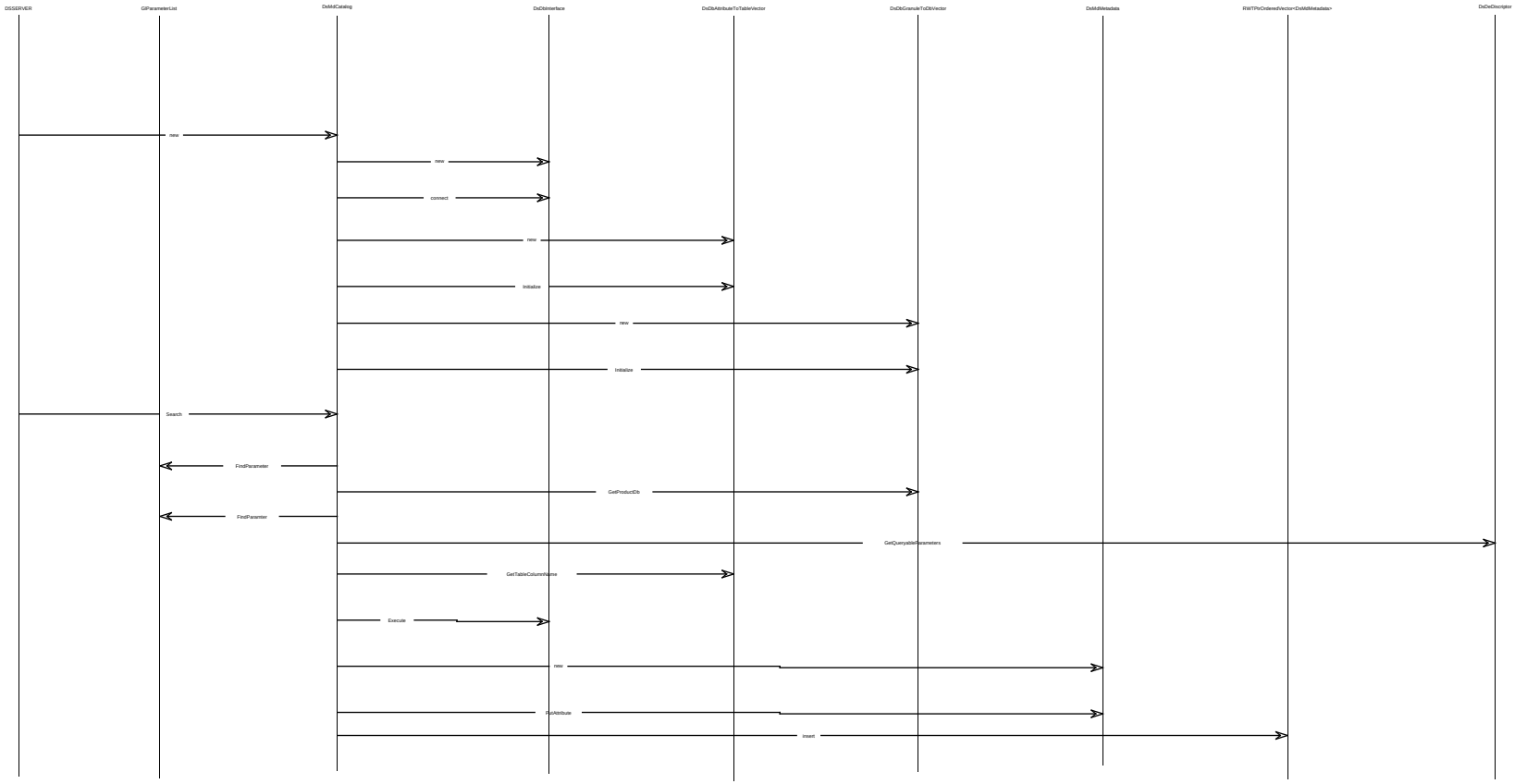


Figure 4.4-8. SDSRV_Catalog_Search Dynamic Model

4.4.9 SDSRV_Catalog_Updating_Metadata

Summary

This scenario shows the sequence of events to update attributes of an existing metadata object and make the change persistent in the SDSRV databases. The metadata object is first updated by the initiator using DsMdMetadata operations. The instance of the DsMdMetata class is then passed to the DsMdCatalog to make the update permanent in the database. The scenario shows the events used by the DsMdCatalog class to establish the database context through the DsDbInterface class, to perform attributes to databases, tables, columns mapping through database DsDbWrapper classes (DsDbAttributeToTableVector and DsDbGranuleToDbVector) and to execute a SQL update command using the DsDbInterface class.

Assumptions

- The DsMdMetadata object being updated should already exist before the update operation. This scenario also assumes that the DsMdCatalog object does not exist before the update operation. Otherwise the construction part of the DsMdCatalog object and the subsequent initialization of other database wrapper objects will be skipped. In which case, a direct UpdateGranuleMetadata is invoked using the DsMdCatalog class. It is preferable for performance reason to avoid reestablishing database connections for each single update.
- The update is persistent. Any subsequent retrieval of the metadata object (either by the same user or other user) will see the effect of the update.

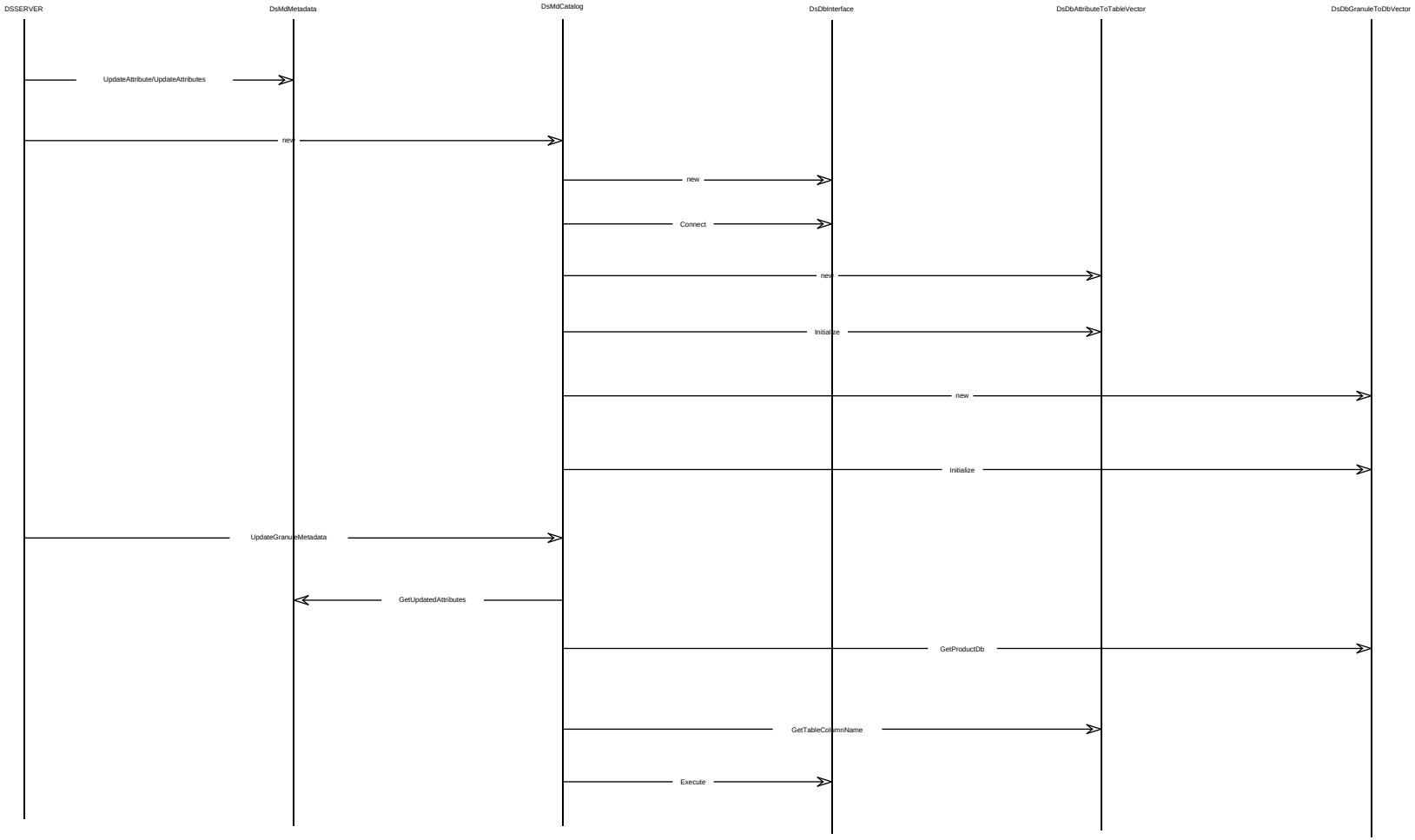


Figure 4.4-9. SDSRV_Catalog_Updating_Metadata Dynamic Model

4.4.10 SDSRV_Changing_A_Request_Priority

Summary

This scenario shows how a client changes the priority of a request that has been submitted to the server but is not yet executing. The two DsClSubmittedRequest classes in the diagram represent the client-side and server-side of the distributed DsClSubmittedRequest object. Changing the priority of a request simply requires that the priority field in the server-side request information is updated, since the code that gets the next request to execute searches the vector for the highest-priority request every time. Note that if the request was executing, the server-side DsClSubmittedRequest would return a fail status.

Assumptions

- the client has all necessary permissions - the request is queued on the server
- all steps execute successfully

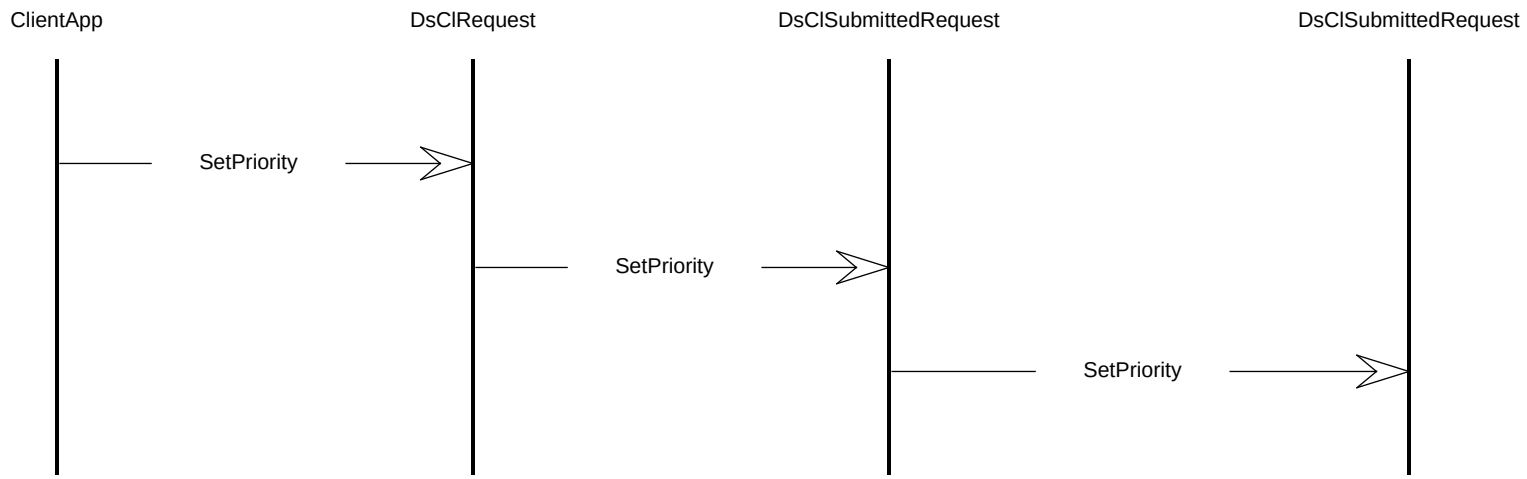


Figure 4.4-10. SDSRV_Changing_A_Request_Priority Dynamic Model

4.4.11 SDSRV_Client_Browsing

Summary

This scenario shows how the client classes are used to browse an item from a data server. The client is assumed to have created an DsClESDTReferenceCollector connected to the desired server, and is required to construct a DsClRequest containing the proper browse command. All of the mechanics leading up to obtaining results are given in the SDSRV_Client_Request_Submission scenario. The results are in a 3-level GlParameterList.

The top-level list is the results from executing the entire request. It is composed of a series of GlParameterList(s), one for each command within the request. (Of course, in this case, there's only one command in the request). Each of these command GlParameterList(s) is also composed of a series of GlParameterList(s), one for each ESDT currently in the collection. (Again, this scenario assumes only one ESDT in the collection). Given this structure, the first "at(0)" retrieves the results for the command, and the second "at(0)" retrieves the results for the ESDT. This is a parameter of type GlBinaryP which contains the actual browse data. Note that there could be several parameters in this final list that give, for example, the UR of ESDT from which the browse data came, or perhaps its size. If these values were provided, the parameters would be named and typed appropriately, and the client could sift through the return parameters to find the ones that are interesting.

Assumptions

- user has logged on and has proper permissions
- the steps in the SDSRV_Client_Request_Submission scenario occur but are not shown
- DsClESDTReferenceCollector is already created and populated with one ESDT
- the request will contain exactly one command

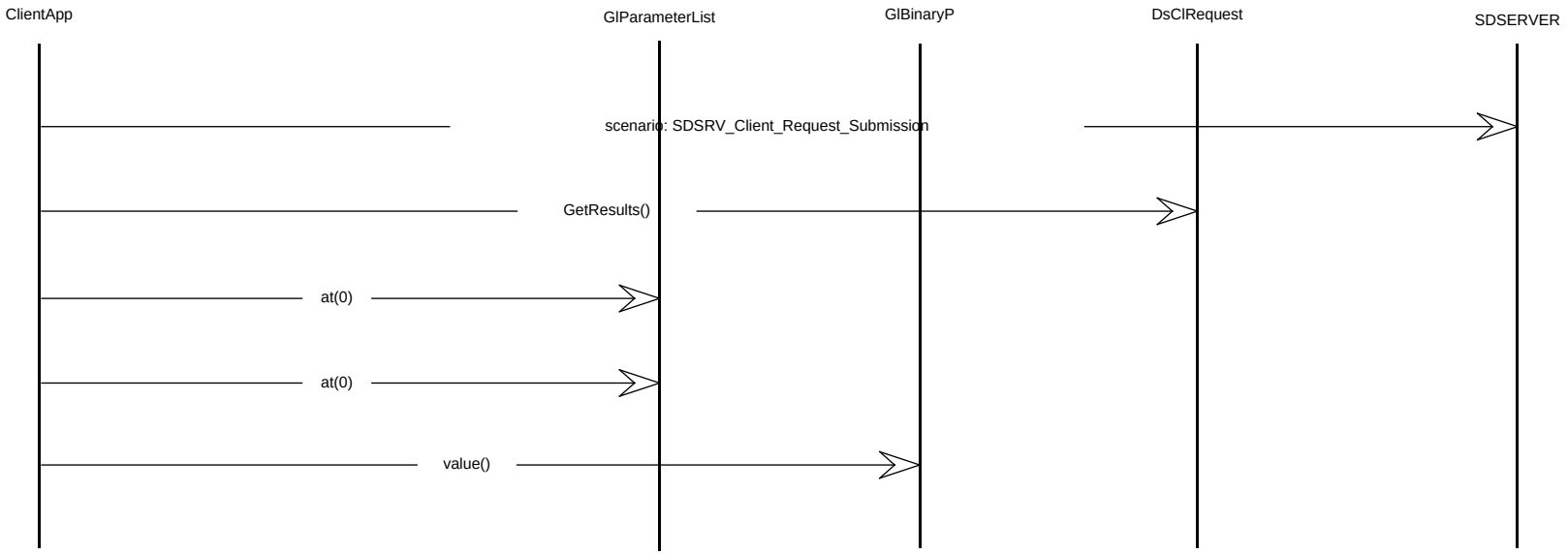


Figure 4.4-11. SDSRV_Client_Browsing Dynamic Model

4.4.12 SDSRV_Client_Connecting_to_a_Data_Server

Summary

The ECSNETWORK line in the event trace shows where the various objects exist with respect to the platforms in the system. The DsClESDTReferenceCollector object is a distributed object on the client side. When the client software creates the DsClESDTReferenceConnection on the server side, the DsSrSession object is created on the server side. This is done via OODCE calls to the Directory Naming Service and the DsFactory object via RPCs. Once the DsSrSession object has been created, it registers to the DsSrServer object, which exists the entire time that the Dataserver is running. Registering with the DsSrServer object provides the DsSrSession object with knowledge of the internal structure of the Dataserver (address of well-known server processes, such as DsSrSubscriptionInterface or DsSrSubmittedRequestVector). Once the DsSrSession has been registered, it creates an empty DsSrWorkingCollection and an empty DsSrRequestVector to handle future requests. When the DsSrSession is all set, it returns the ID which has been assigned to it by the DsSrServer to the client DsClESDTReferenceCollector. The DsClESDTReferenceCollector stores the ConnectionID for internal processing and also returns it to the client software so that the session can be resumed if it is interrupted or suspended. (This last part is not Release A.)

Assumptions

- Client software knows the address of the dataserver to which to connect
- Client software has already collected all user information (i.e., has a MSSUserProfile object available)
- Client software wishes to create the connection directly

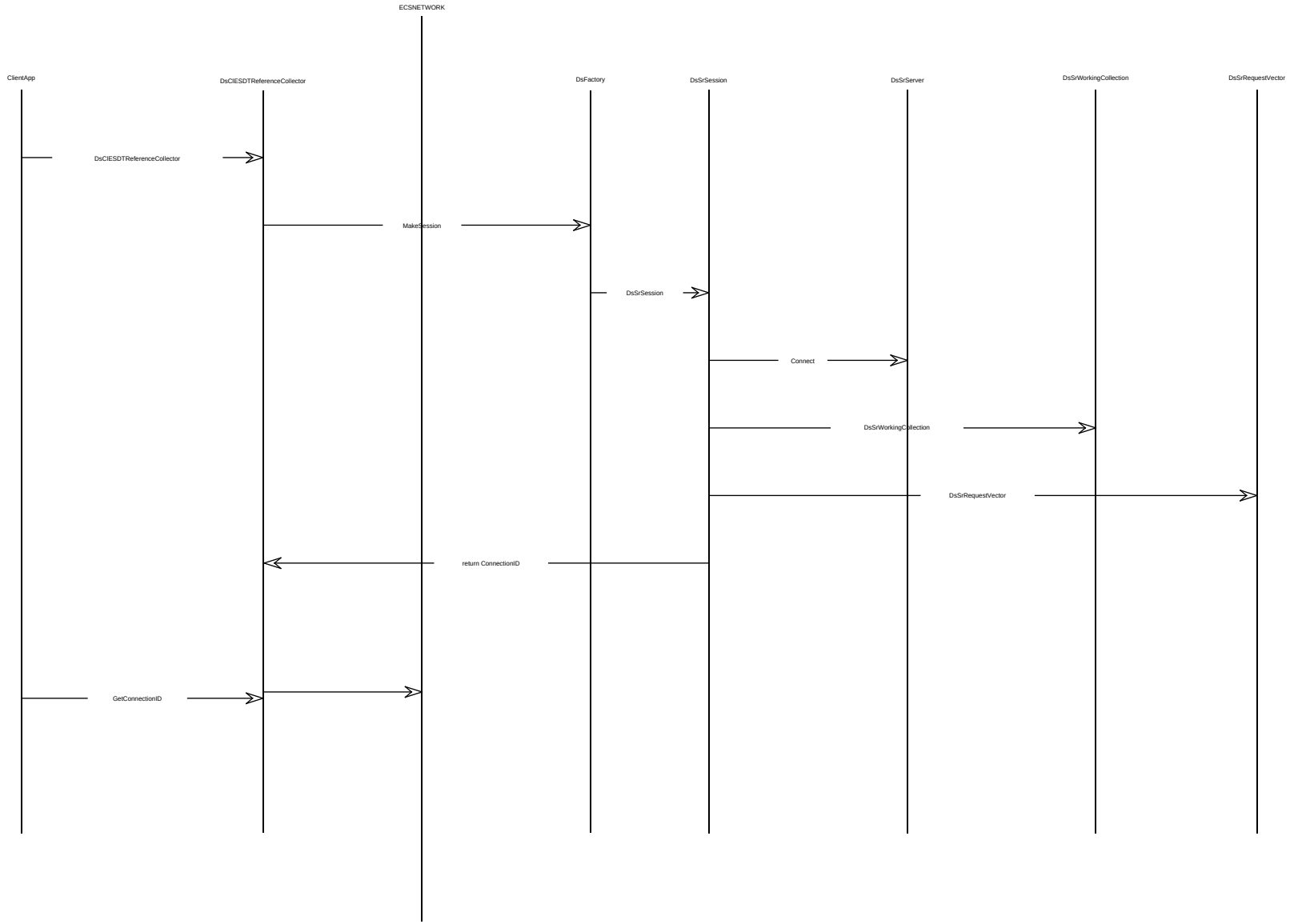


Figure 4.4-12. SDSRV_Client_Connecting_to_a_Data_Server Dynamic Model

4.4.13 SDSRV_Client_Request_Submission

Summary

This scenario shows how the client classes handle a request. The SDSRV_Asynchronous_Status_Updates scenario shows the details of the request's connection to the data server; this scenario shows the steps used to submit it and how results are retrieved. The DsClESDTReferenceCollector is assumed to be already created and connected, so the client is only required to construct a request and submit it. The client regains

control when the status changes; this scenario shows only the return when status is DONE. The client is then able to retrieve the results and clean up the objects.

Assumptions

- user has logged on and has proper permissions - DsClESDTReferenceCollector is already created
- the steps in the SDSRV_Asynchronous_Status_Updates scenario occur but are not shown
- the request contains exactly one command - all steps execute successfully

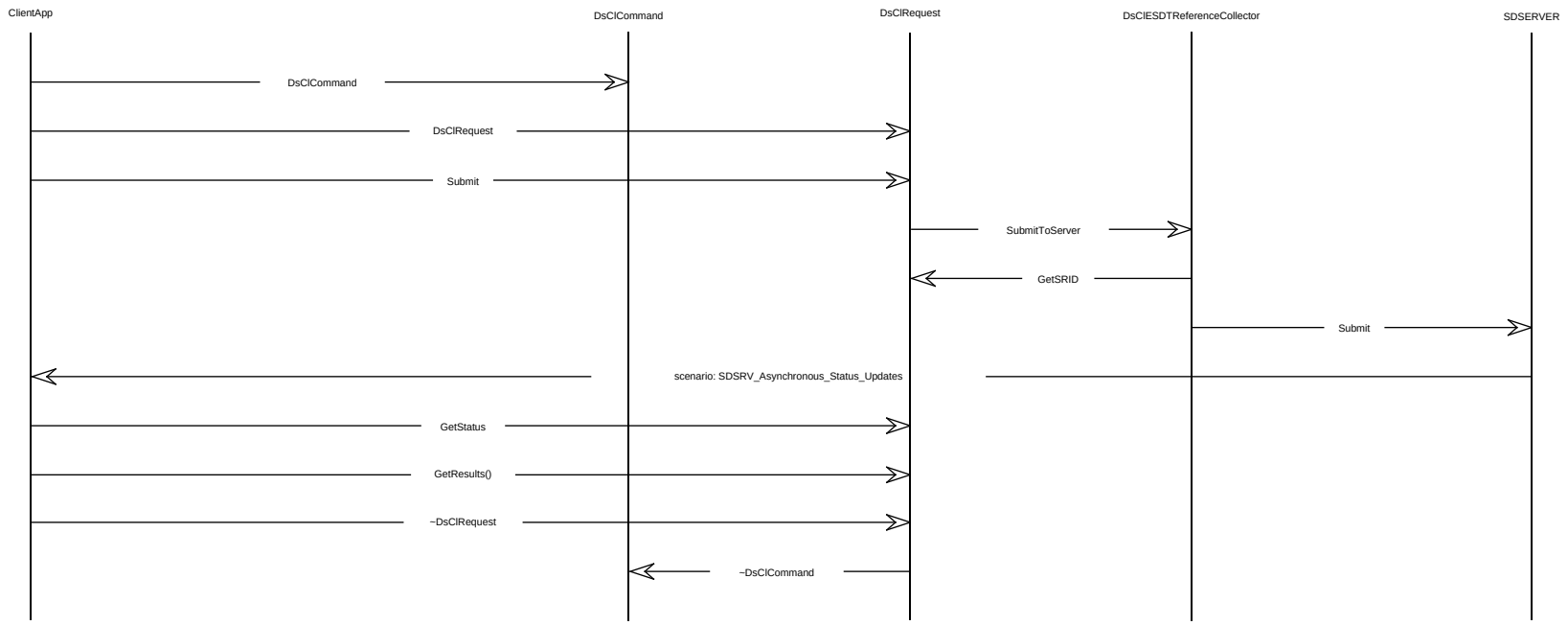


Figure 4.4-13. SDSRV_Client_Request_Submission Dynamic Model

4.4.14 SDSRV_Client_Searching

Summary

The client software creates a DsClQuery object. This is a container object which holds values for parts of a query (e.g. - projection and constraints). The "from clause" of the query is presumed to be defined by the data server to which the query is sent. That is, the dataserver does not require table names to be provided; the data server catalog contains all the information necessary to locate the metadata based on the constraints. Once the DsClQuery has been created, two attributes must be set, the callback and the constraints. The client software creates a GlCallback object and sets the DsClQuery object's Callback attribute to it. This callback is the local portion of the end-to-end asynchronous message passing mechanism. This callback is a standard callback which is invoked by the DsClESDTReferenceCollector once it (the DsClESDTReferenceCollector) has received and processed the results of a user request. The second attribute of DsClQuery, (Constraints), takes an already-packaged GlParameterList of search attributes and the values to be used for them. Once the DsClQuery object attributes have been set, the client software sends the DsClQuery object to the DsClESDTReferenceCollector as a parameter in the Search method. The DsClESDTReferenceCollector calls the DsClQuery's conversion method to formulate a data server command, which it then uses to construct a DsClRequest object. Once the request is created, it's Submit method is invoked. The processing of the DsClRequest is documented in Scenario SDSRV_Client_Request_Submission. The handling of the remote request status is described in Scenario SDSRV_Asynchronous_Status_Updates. Once the request has completed (status has been set to "done" and the local DsClRequest callback has been invoked), the DsClESDTReferenceCollector retrieves the results from the DsClRequest object. The results are a set of commands. The DsClESDTReferenceCollector UpdateState method iterates over the set of commands, executing each in turn. In this example, all of the commands will be "add" commands (because the working collection was empty prior to the search). The result of the command processing is to make the DsClESDTReferenceCollector on the client-side match the DsSrWorkingCollection on the server side. Once all the commands have been processed, the DsClQuery object's (local) callback is invoked to alert the client software that the request has completed. The client software uses standard vector commands to iterate over the set of DsClESDTReferences. At any point in time after the DsClQuery callback has been invoked with a status of DONE, the DsClQuery object can be deleted.

Assumptions

- Client software has already established a connection to the Data Server (see Scenario SDSRV_Client_Connecting_to_a_Data_Server)
- Client software has already collected query constraints from the user
- Client software has already converted query constraints to a GlParameterList
- All parameters in the GlParameterList are valid queryable parameters

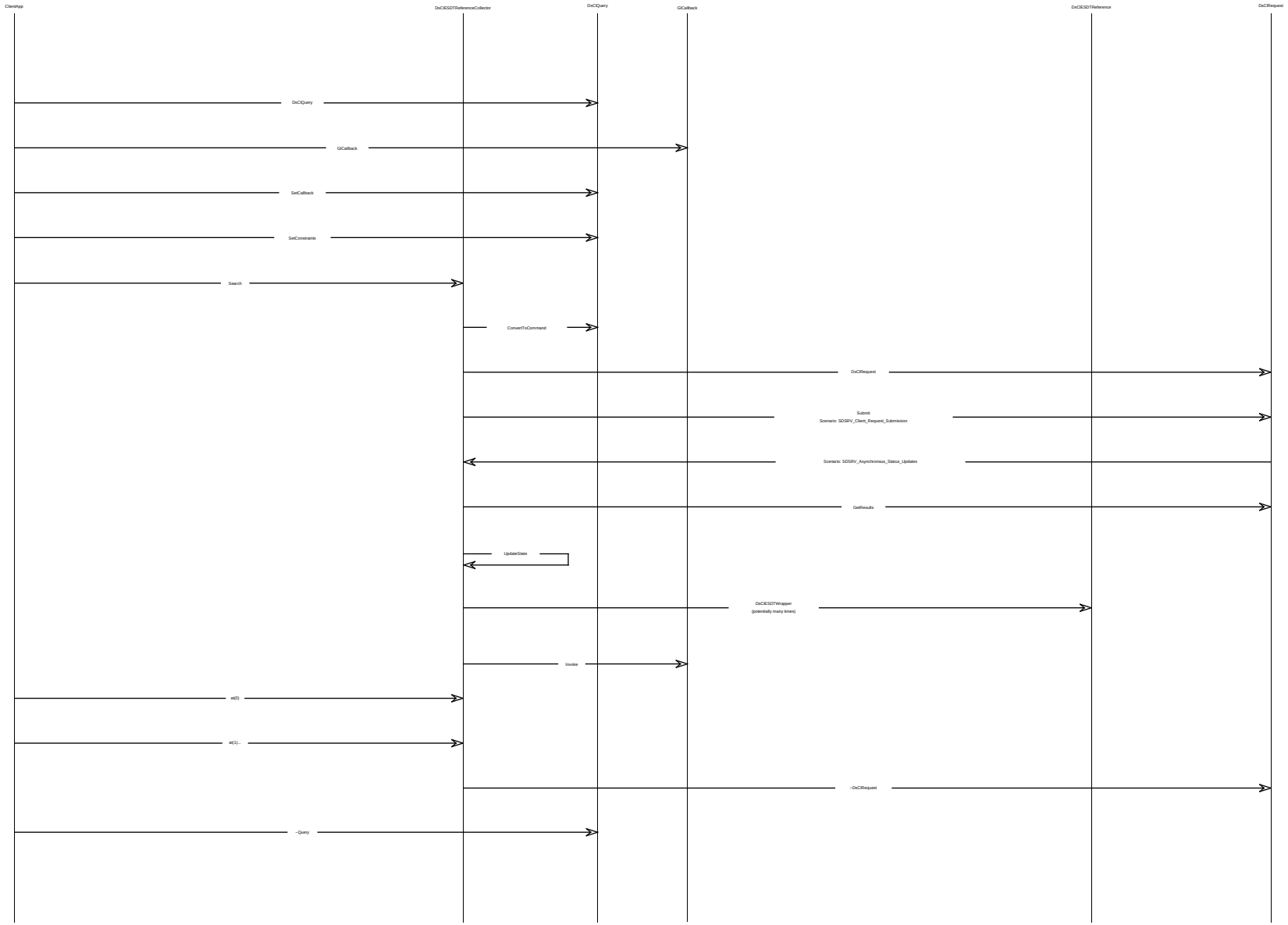


Figure 4.4-14. SDSRV_Client_Searching Dynamic Model

4.4.15 SDSRV_Deleting_A_Queued_Request

Summary

This scenario shows the interaction between the client and server when a client cancels and deletes a request that is queued on the server. Since the client creates and controls his DsClRequest objects, he can simply call Cancel on the desired DsClRequest. This is passed through the DsClESDTReferenceCollector distributed object to the server (DsSrSession), where the request is cancelled. This returns a CANCEL status to the client through the SDSRV_Asynchronous_Status_Updates scenario, which indicates to the client that the DsClRequest may be deleted.

Assumptions

- the client has all necessary permissions - the request is queued on the server
- all steps execute successfully

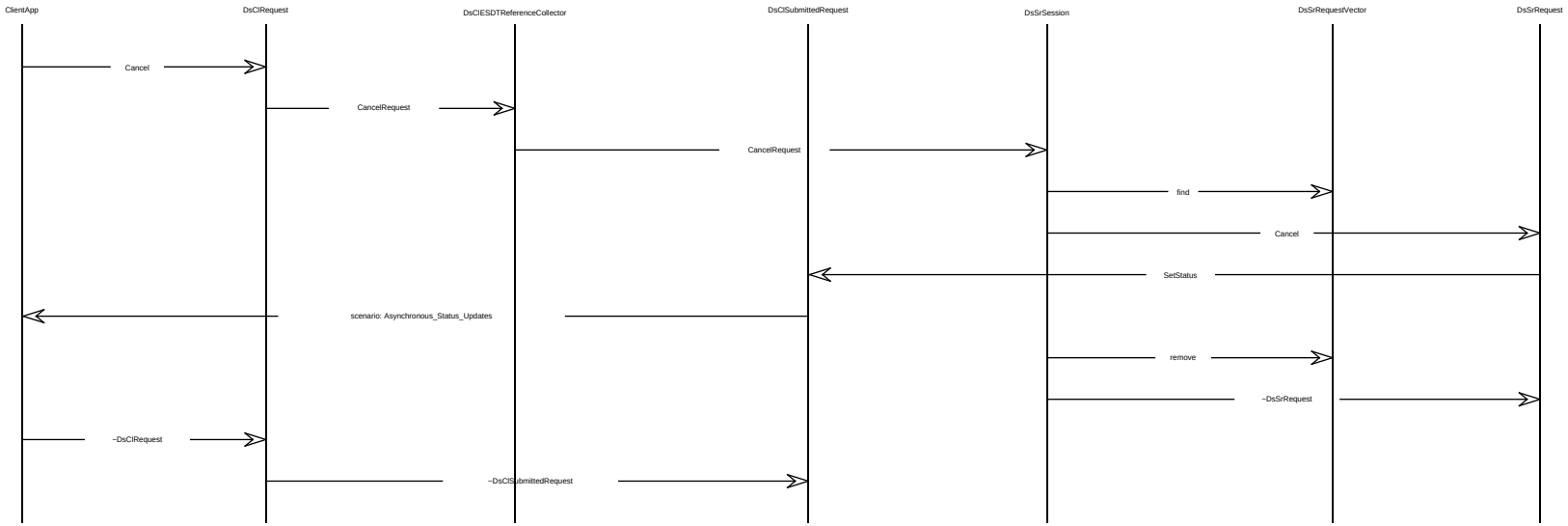


Figure 4.4-15. SDSRV_Deleting_A_Queued_Request Dynamic Model

4.4.16 SDSRV_Ending_Session_No_Active_Request

Summary

The ECSNETWORK line in the event trace shows where the various objects exist with respect to the platforms in the system. The DsClESDTReferenceCollector object is a distributed object. When the client software removes the DsClESDTReferenceCollector, that object must clean up on both the client side and the server side. On the client side of the network, the DsClESDTReferenceCollector removes the proxy objects DsClTypeInfo (potentially more than one), and DsClESDTReference (potentially more than one). Because the DsClESDTReferenceCollector is a distributed object, the corresponding server-side object (DsSrSession) is also removed. The first thing that the DsSession does is unregister from the DsSrServer object. This removes the DsSession from being considered "active" in the data server system. After it is no longer registered, the DsSession object removes its DsSrWorkingCollection (which in turn deletes all associated DsGeESDTWrappers) and its (empty) DsSrRequestVector.

Assumptions

- Client software has already executed a request, and the request has finished
- The WorkingCollection is holding some ESDT objects.
- Client software has previously cleaned up all no-longer-useful objects (i.e., if a previous request was a query, the DsClQuery object and the DsClRequest has already been removed from memory)

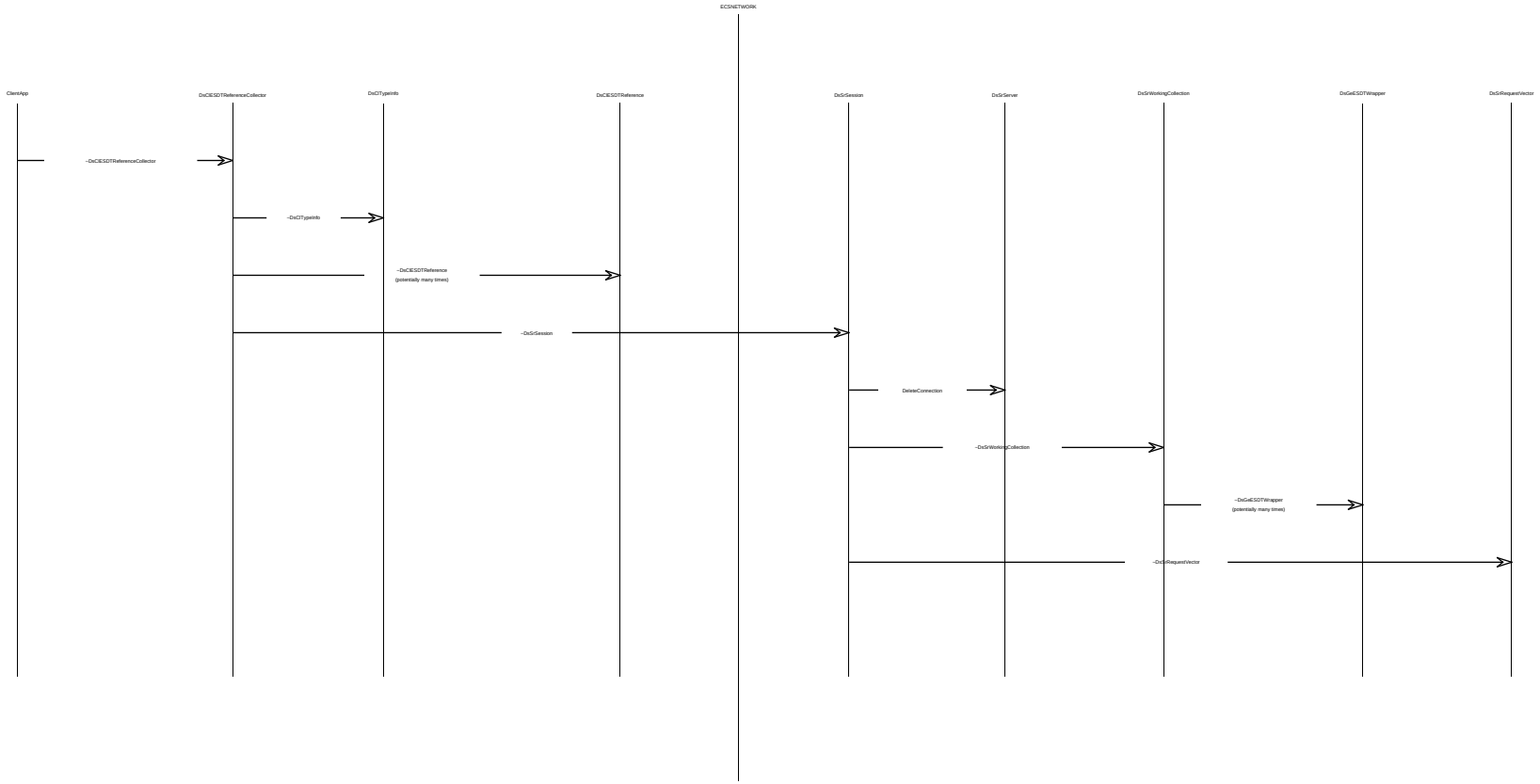


Figure 4.4-16. SDSRV_Ending_Session_No_Active_Request Dynamic Model

4.4.17 SDSRV_Fulfilling_a_One-time_Subscription

Summary

This scenario shows how the SDSRV fulfills a previously submitted subscription. In this scenario the originating object has detected the advertised and subscribed event. It then triggers that event for processing within the Subscription Server. The subscription example illustrated here is a one-time subscription, meaning after one time the event detection is processed, the subscription will be removed as an active subscription.

Assumptions

- The submitted subscription is for a simple notification to be sent.

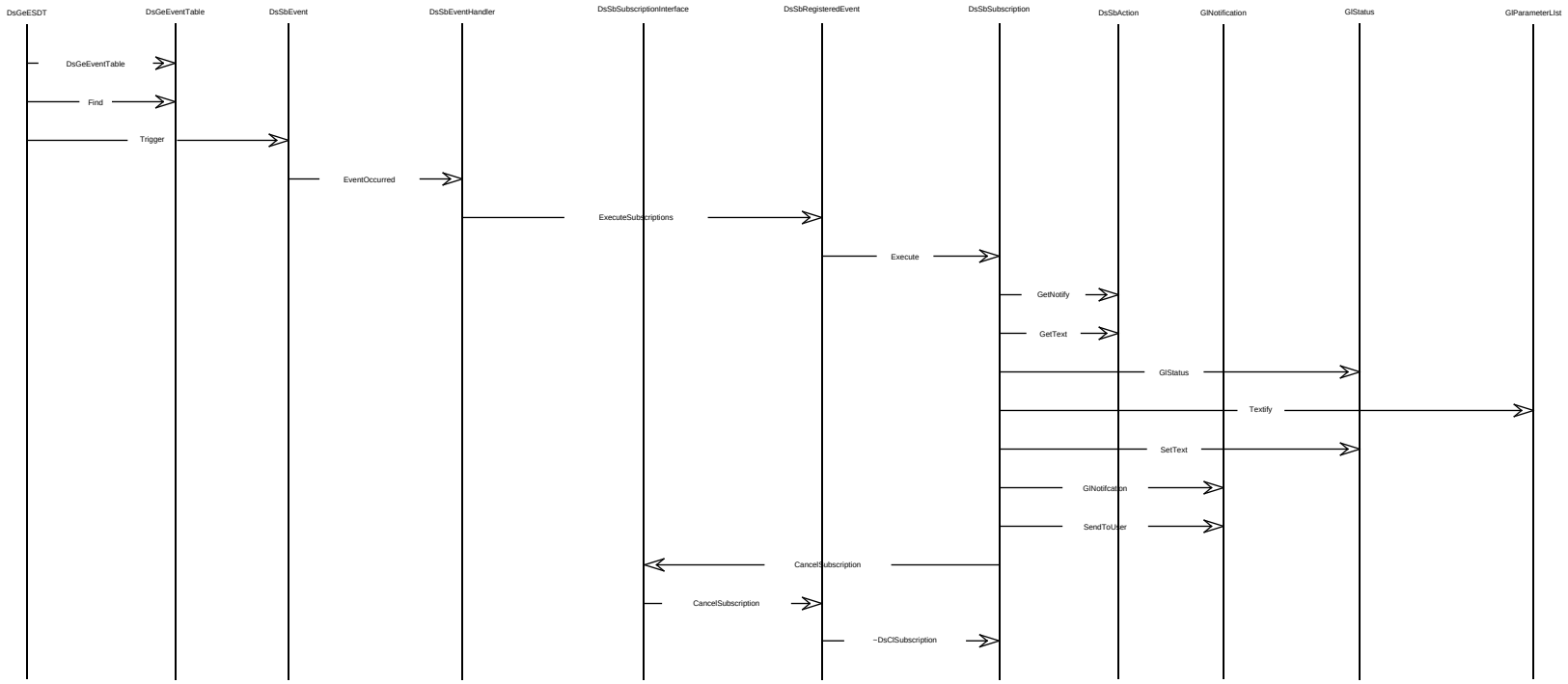


Figure 4.4-17. SDSRV_Fulfilling_a_One-time_Subscription Dynamic Model

4.4.18 SDSRV_Fulfilling_Open_Ended_Subscription

Summary

This scenario shows how the SDSRV fulfills a previously submitted subscription. In this scenario the originating object has detected the advertised and subscribed event. It then triggers that event for processing within the Subscription Server. The subscription example illustrated here is a open-ended subscription, meaning it will continue to be active, waiting for future occurrences of the event.

Assumptions

- The submitted subscription is for a simple notification to be sent.

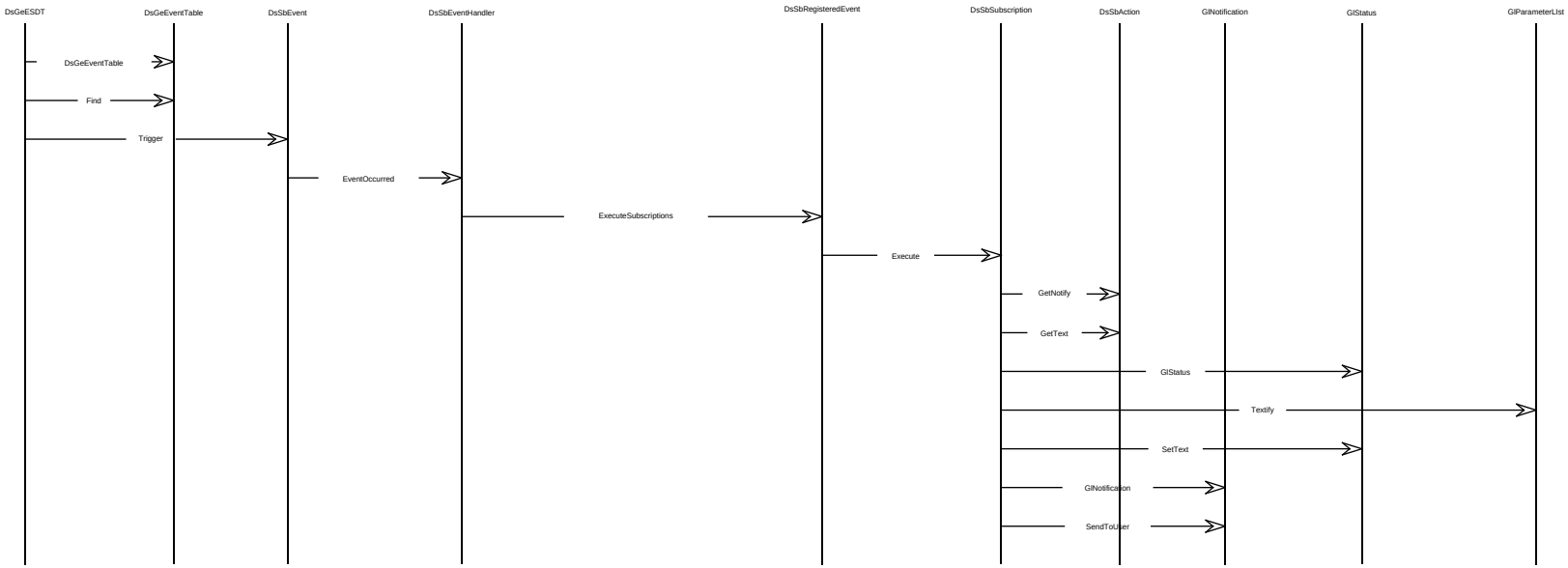


Figure 4.4-18. SDSRV_Fulfilling_Open_Ended_Subscription Dynamic Model

4.4.19 SDSRV_Inserting_Composite_ESDT

Summary

This scenario begins at the point where the server session receives an Insert command request from the client (refer to scenario: SDSRV_Server_Request_Handling). The client for this scenario is the Ingest subsystem. The session determines that the already established working collection is required to service the command. The working collection instantiates an 'empty' ESDT of the correct type using dynamic binding (see scenario: SDSRV_Instantiating_an_ESDT). For each composite granule represented as files within the command the 'Internalize' command is called (in this scenario a browse ESDT granule). Each ESDT internalize returns a vector of populated metadata objects and filenames. Three main operations are then performed by the 'anchor' ESDT object (DsViVIRS1B). First the science data file(s) are sent to the archive for storage, second the filenames and unique granule IDs are added to the granule metadata (and thus the product and granule association is maintained) and third the metadata is written to the database.

Points to note:

The unique granule ID is set during insertion of the granule metadata. The ID is passed back to the working collection as the result of a successful Externalize operation. Ultimately this ID will be returned to the client. The associations between granules inserted in this manner are maintained by adding the unique granule ID's into the relevant metadata streams sent to the database.

Assumptions

- Ingest supplies a validated file containing P=V metadata values
- Ingest supplies a file containing scientific data
- Ingest supplies a file containing related browse data.
- A work area has already been assigned
- A connection has already been established to the dataserver
- A working collection already exists
- Data has been staged ready for insertion into the database
- The Insert request is as a result of call from the Ingest Subsystem
- Metadata stream has already been validated
- Ingest supplies a validated file containing P=V metadata values
- Ingest supplies a file containing scientific data
- A work area has already been assigned
- A connection has already been established to the dataserver
- A working collection already exists
- Data has been staged ready for insertion into the database
- The Insert request is as a result of call from the Ingest Subsystem
- Metadata stream has already been validated

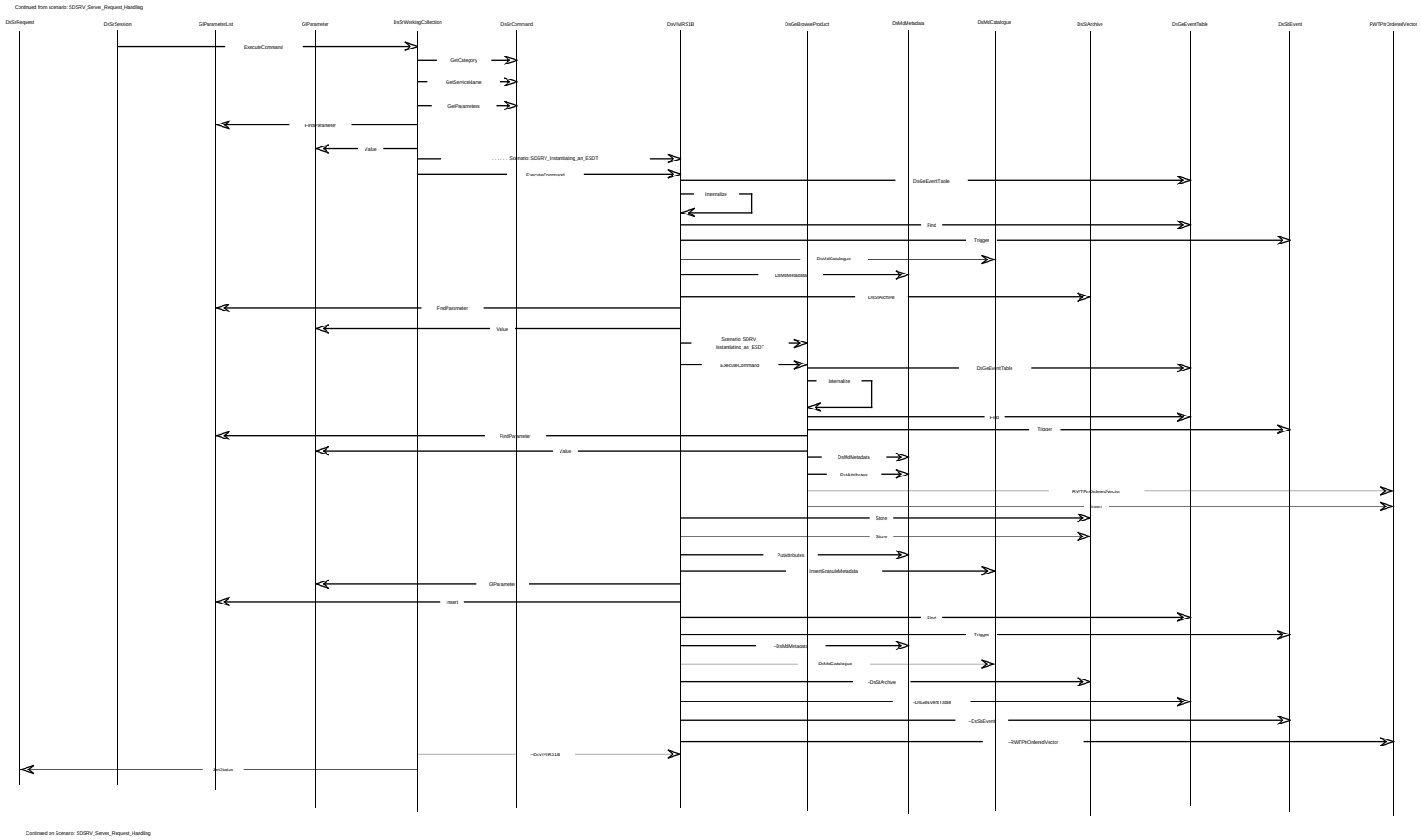


Figure 4.4-19. SDSRV_Inserting_Composite_ESDT Dynamic Model

4.4.20 SDSRV_Inserting_New_ESDT

Summary

The purpose of this scenario is to show how a new Data Type is incorporated into the currently executing Science Data Server. The Data Server Administrator initiates this scenario from the DsGuAdminGUI. This scenario begins after the administrator has selected the option to Add New DataType, has supplied the required parameters for this option, and has selected Execute. The required parameters for adding a new Data Type are the new Data Type's name and version number, a file which is the dynamic linked library (DLL) containing its implementation and an ASCII file containing the ESDT Descriptor information.

Upon receiving this Administrators request, the SDSRV performs a 3-step process for adding a new Data Type which is controlled by the DsGeESDTAdmin object. First, the new type is registered as a known Data Type within this SDSRV. Verification that the type is new is performed and a unique identifier is assigned to this type. This information is persistently stored which establishes this Data Type's existence.

Second, the given DLL filename is associated with this newly established Data Type. This association is persistently stored which enables the Data Type's implementation to be used for performing services of this new type.

Finally, the new ESDT Descriptor information is added to the set of descriptors that this SDSRV knows about. The ESDT Descriptor file contains well-defined groups of information such as the Core Metadata Group. The file is parsed to extract and store the individual groups persistently as separately accessible entities. Once the file has been processed, the ESDT Descriptor initialization process is begun. This process advertises the services that this Data Type provides, registers events that will be reported (for triggering subscriptions), and exports data dictionary information to the Data Management Subsystem.

Assumptions

- The Data Type does not already exist.
- The DLL file exists on the same host as the SDSRV.
- The Descriptor file exists and has the appropriate format.

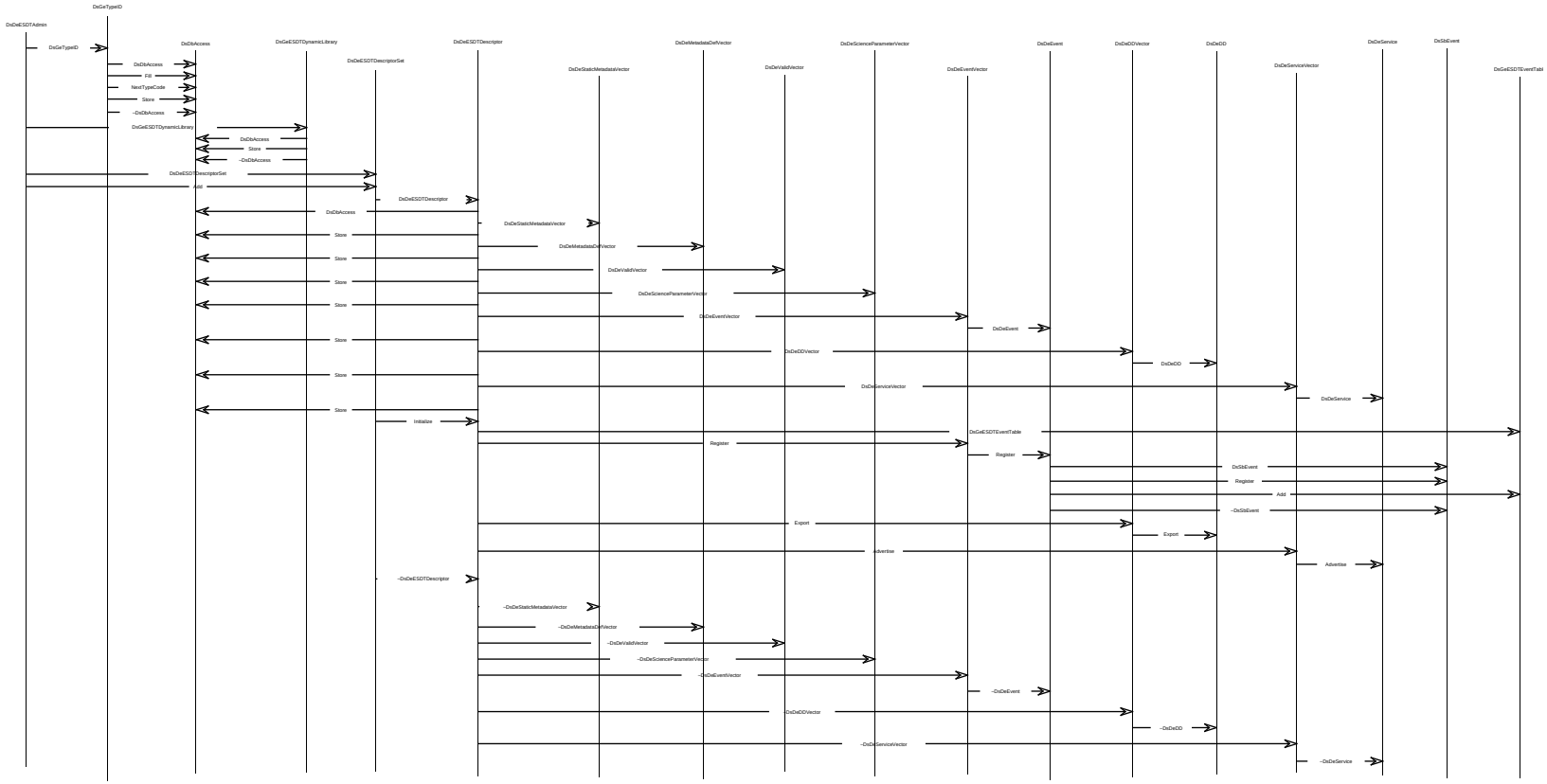


Figure 4.4-20. SDSRV_Inserting_New_ESDT Dynamic Model

4.4.21 SDSRV_Inserting_Single_ESDT

Summary

This scenario begins at the point where the server session receives an Insert command request from the client (refer to scenario: SDSRV_Server_Request_Handling). The client for this scenario is the Ingest subsystem. The session determines that the already established working collection is required to service the command. The working collection instantiates an 'empty' ESDT of the correct type using dynamic binding (see scenario: SDSRV_Instantiating_an_ESDT). Three main operations are then performed. First the science data file(s) are sent to the archive for storage, second the filenames and unique granule ID are added to the granule metadata and third the metadata is written to the database.

Points to note:

The unique granule ID is set during insertion of the granule metadata. The ID is passed back to the working collection as the result of a successful Externalize operation. Ultimately this ID will be returned to the client.

Assumptions

- Ingest supplies a validated file containing P=V metadata values
- Ingest supplies a file containing scientific data
- A work area has already been assigned
- A connection has already been established to the dataserver
- A working collection already exists
- Data has been staged ready for insertion into the database
- The Insert request is as a result of call from the Ingest Subsystem
- Metadata stream has already been validated

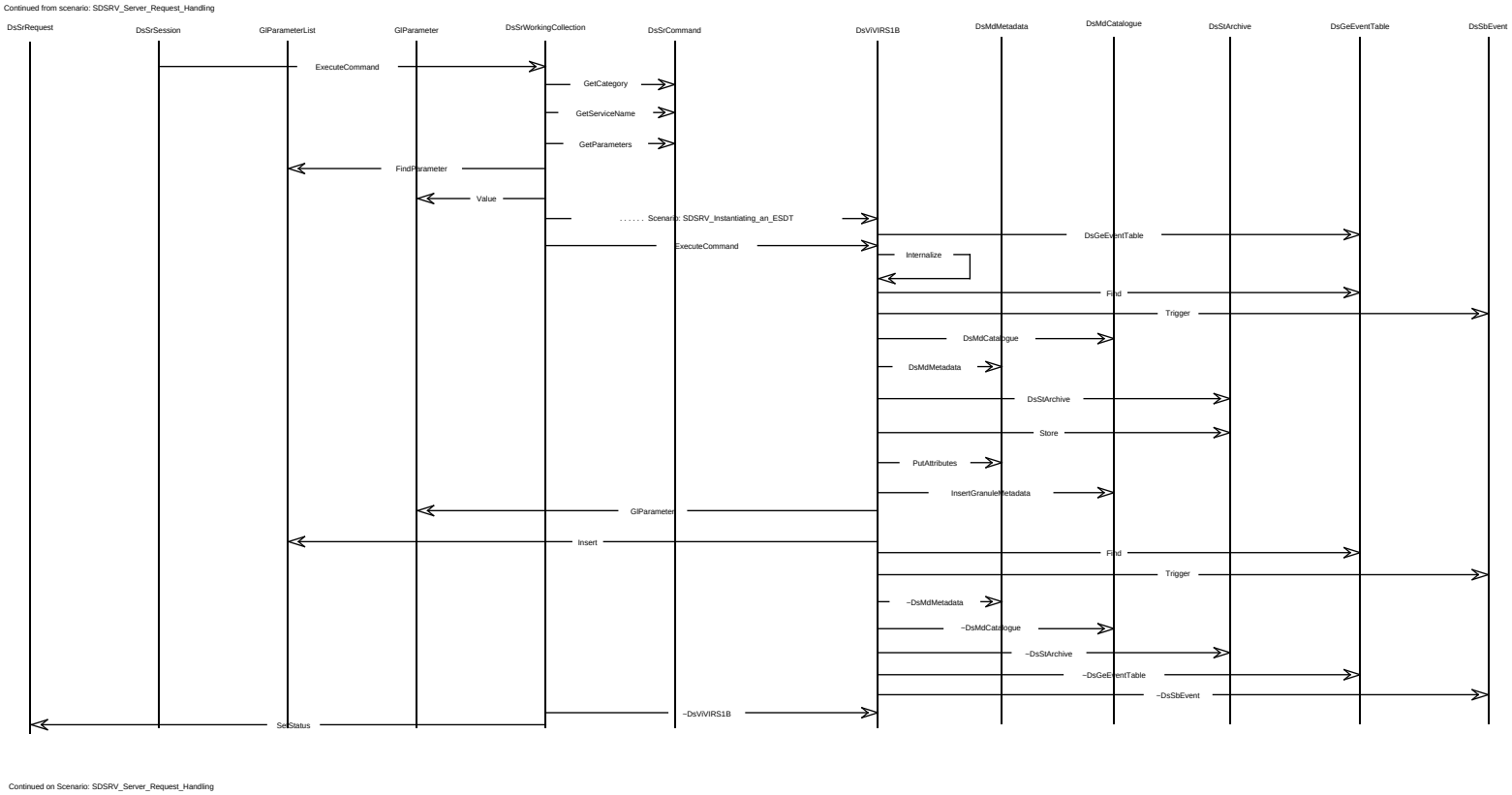


Figure 4.4-21. SDSRV_Inserting_Single_ESDT Dynamic Model

4.4.22 SDSRV_Instantiating_an_ESDT

Summary

The scenario begins with the working collection determining what data type is to be instantiated. The ESDTwrapper makes a call to the DsGeDynaminObject, including in that call the data type required. Using appropriate lookup tables the relevant library for the data type in question is loaded and the data type independent function newESDT called. This results in the calling of the data type specific constructor, the end of result of which is the passing of an ESDT object handle back to the ESDTWrapper. From this point on the ESDTWrapper can be treated as a specific data type.

Assumptions

- The data type is already defined (see scenario: SDSRV_Creating_a_new_datatype).
- A connection has already been established to the dataserver.
- A working collection already exists.

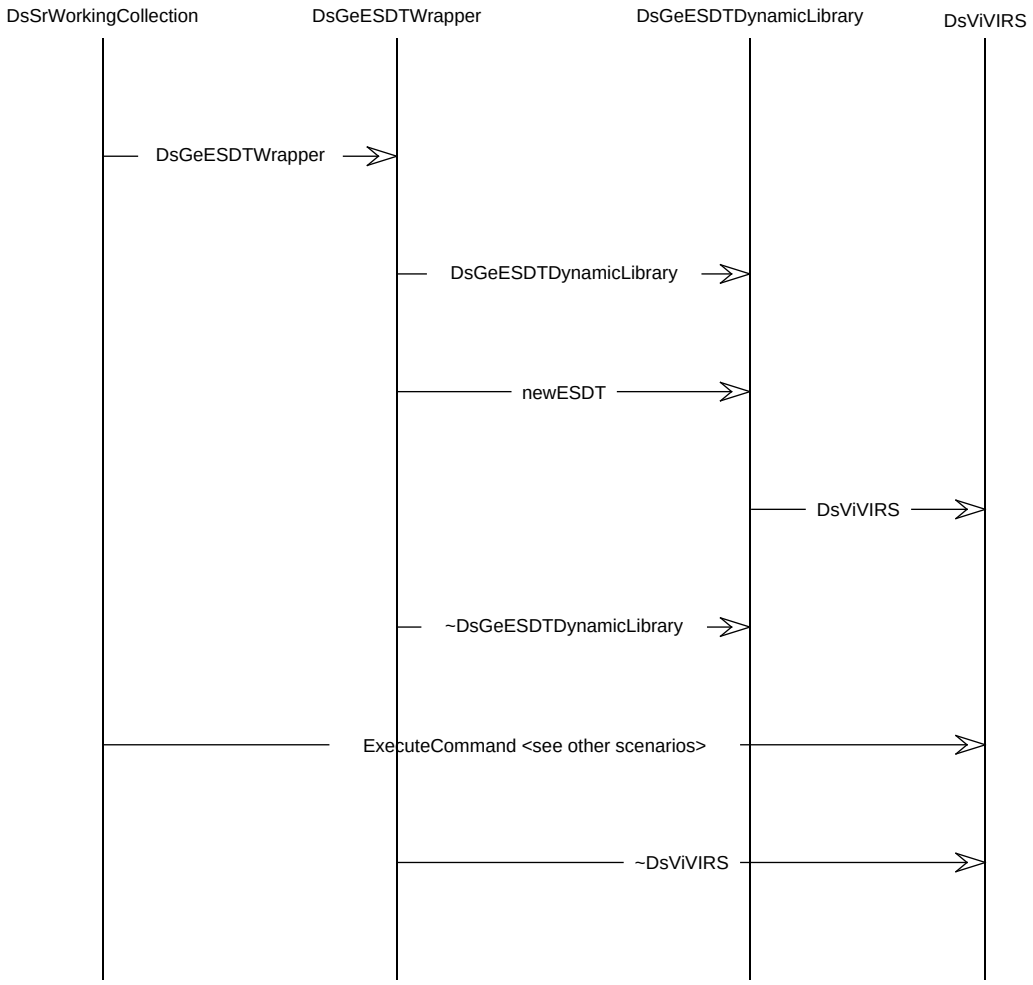


Figure 4.4-22. SDSRV_Instantiating_an_ESDT Dynamic Model

4.4.23 SDSRV_Op_View_Queued_Requests

Summary

This scenario shows how an SDSRV operator's GUI gets the necessary information about currently queued requests from a Science Data Server.

Assumptions

- GUI knows which SDSRV the operator is interested in.
- GUI formats the text representation of the queued requests in a scrollable text widget.

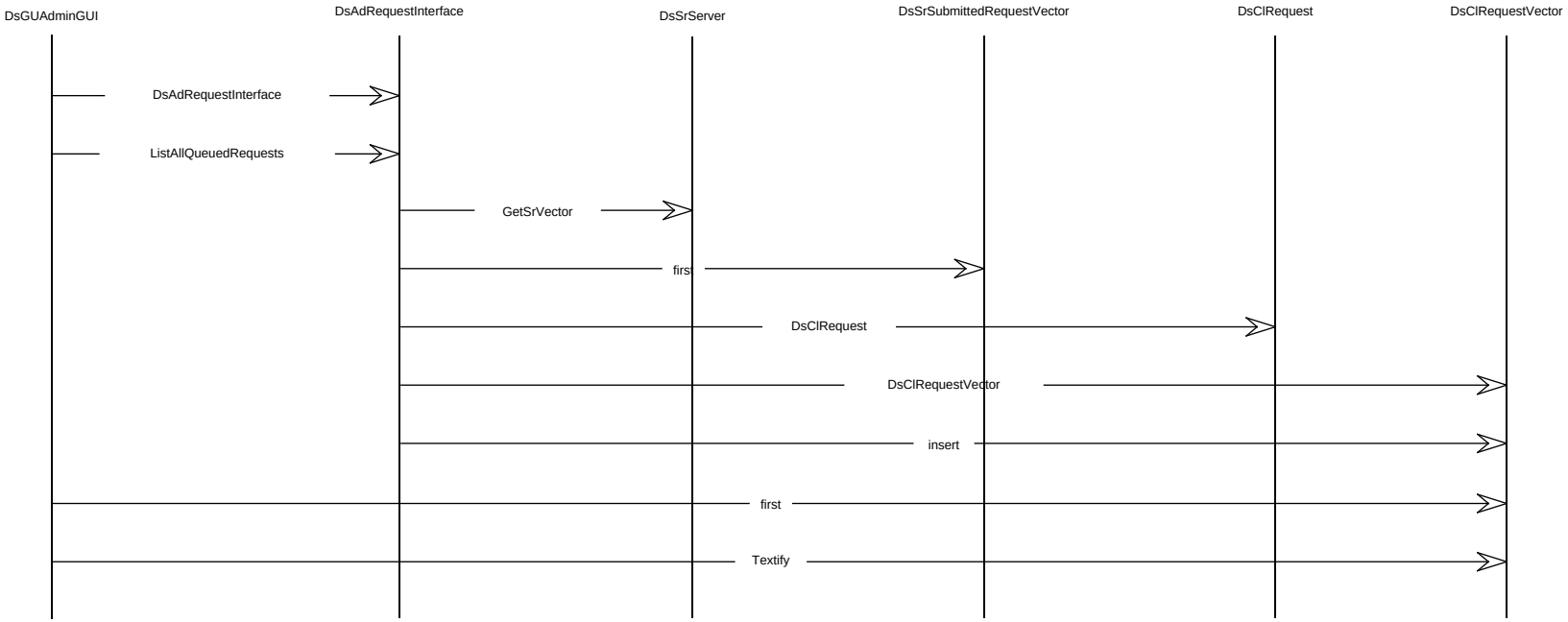


Figure 4.4-23. SDSRV_Op_View_Queued_Requests Dynamic Model

4.4.24 SDSRV_Registering_a_Subscribable_Event

Summary

This purpose of this scenario is to show how a subscribable event gets established. This occurs within the context of a variety of scenarios including adding a new Data Type and Startup of the Science Data Server. Although this scenario focuses on events that are generated by Data Types, this is not meant to imply that all subscribable events are Data Type related. A subscribable event might be the occurrence of a disk being full. This scenario begins after an DsDeESDTDescriptor has been created and it has determined that it must establish its subscribable events.

The DsDeESDTDescriptor establishes a DsGeESDTEventTable for holding all of the events for this Data Type and persistently stores it. Then for an event that the Data Type has the ability to notice, the DsDeESDTDescriptor creates a DsSbEvent with a name, category, and description. Since the DsSbEvent is a distributed object, this causes a creation of an DsSbEvent within the Subscription Server process space. Then the DsDeESDTDescriptor registers the event which notifies the SubscriptionInterface that this is now a known event. The SubscriptionInterface coordinates with the Data Management Subsystem to advertise the fact that this event is subscribable. After the event has been registered, it gets stored persistently in the previously created DsGeESDTEventTable. This process is repeated for each event that this Data Type has the ability to notice.

Assumptions

- The SDSRV has the UR of its Subscription Server Factory.
- The Subscription Server is active.
- There is just one event being registered.
- The DsDeEventVector and DsDeEvent are already created.

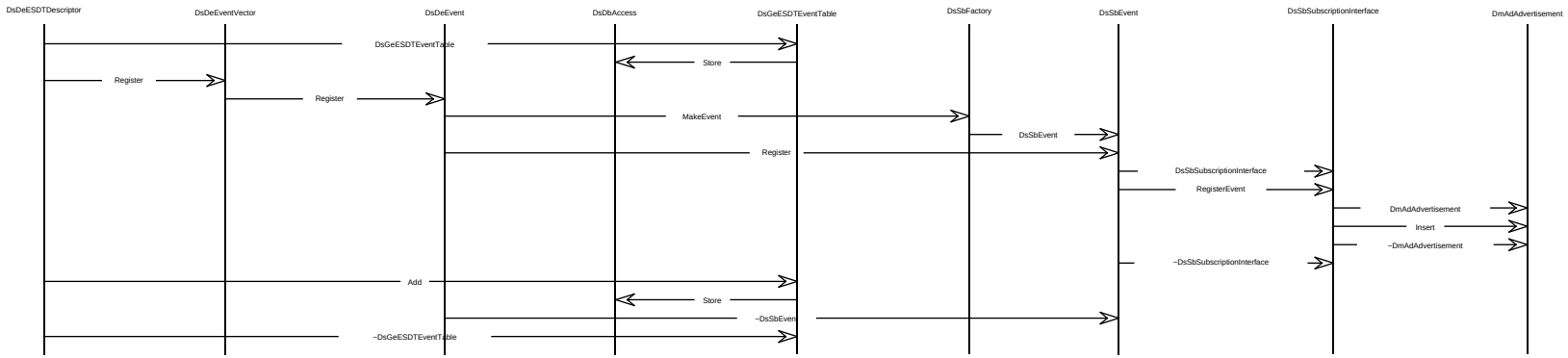


Figure 4.4-24. SDSRV_Registering_a_Subscribable_Event Dynamic Model

4.4.25 SDSRV_Returning_List_of_Subscriptions

Summary

This scenario shows how the DsClSubscriptionCollector is populated with the subscriptions for a specified user. The client tells the DsClSubscriptionCollector (a distributed object) to BuildList, which (server-side) asks the DsSbSubscriptionInterface to build a parameter list of references to the subscriptions for a user. The client-side DsClSubscriptionCollector takes this list and builds a DsClSubscription (only one in this case) from it and adds it to himself.

Assumptions

- There is one subscription for the user
- all steps execute successfully

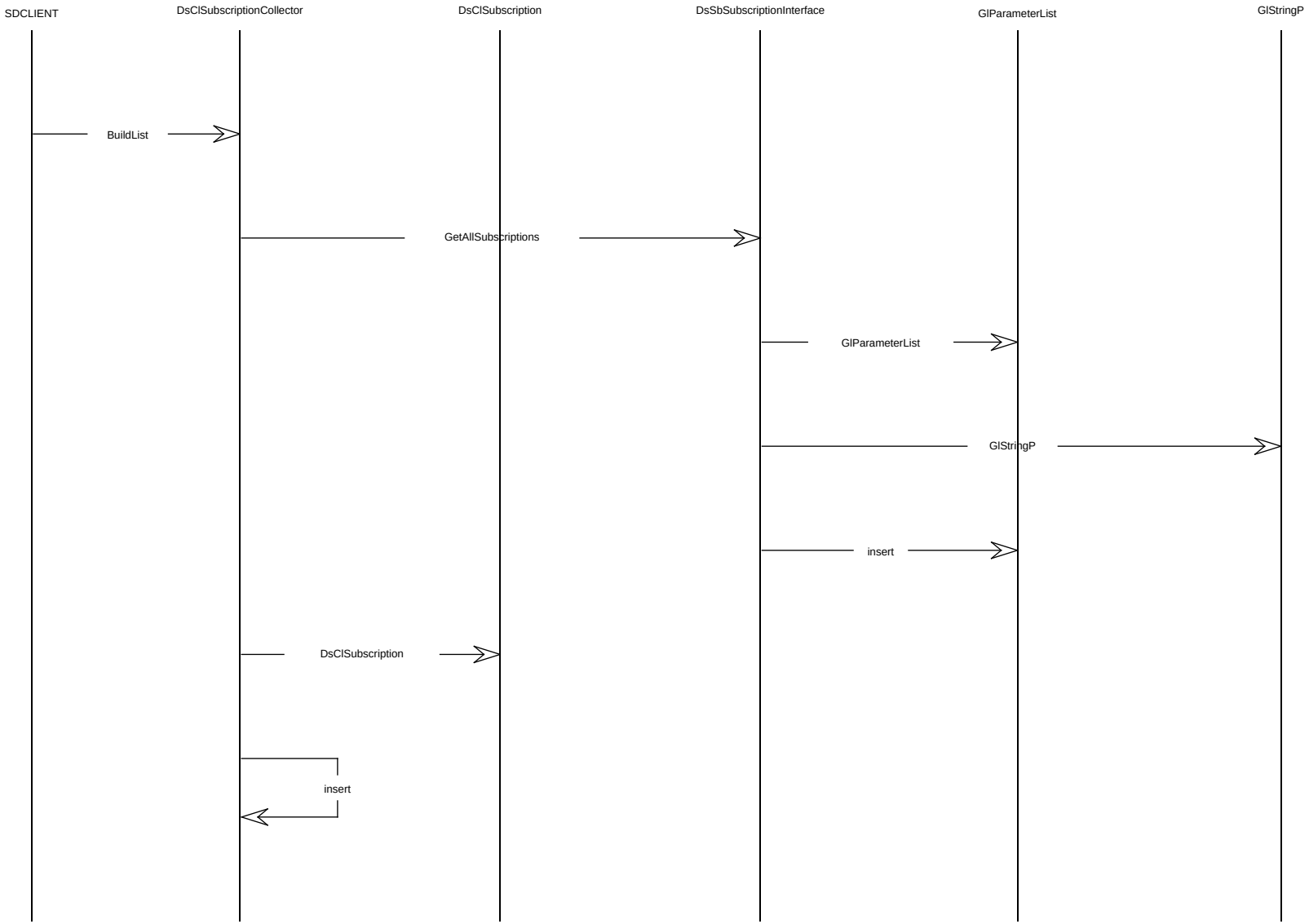


Figure 4.4-25. SDSRV_Returning_List_Of_Subscriptions Dynamic Model

4.4.26 SDSRV_Server_Handling_A_Browse_Request

Summary

This scenario shows the execution of a browse request within the data server. A browse request is really just a request with a single command ("browse") that is executed by an ESDT. The diagram begins with the submission of the request to the DsSrSession, and assumes the steps of the top part of the SDSRV_Server_Request_Handling scenario are executed. The DsSrWorkingCollection is then handed the command to execute, and passes it to the DsGeESDTWrapper. The DsGeESDTWrapper is a class that encapsulates the dynamic binding of ESDT services to the data server process, and passes commands through to the actual ESDT code to execute. The diagram shows the base class "DsGeESDT" executing the command, while in reality a leaf class would perform the actual execution. This abstraction shows that neither the DsSrWorkingCollection nor the DsGeESDTWrapper need to know the exact type of the ESDT, just that it inherits from DsGeESDT. Once the execution of the command has finished, the results are added to the request results, and the remainder of the SDSRV_Server_Request_Handling scenario is finished.

Assumptions

- the DsSrWorkingCollection contains a single ESDT, which is already instantiated
- the request contains exactly one command - the client has all necessary permissions
- all steps execute successfully

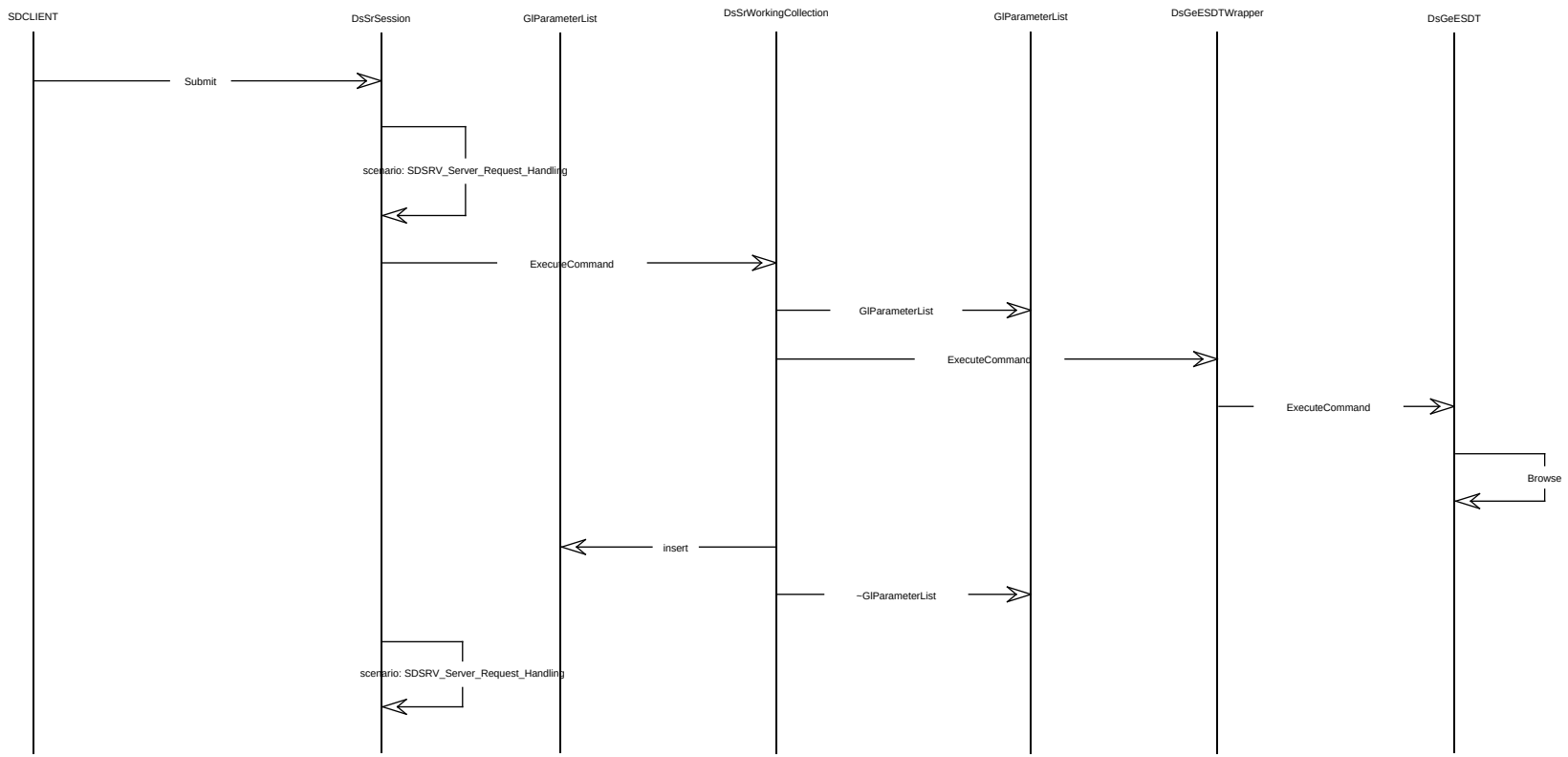


Figure 4.4-26. SDSRV_Server_Handling_A_Browse_Request Dynamic Model

4.4.27 SDSRV_Server_Handling_A_Search_Request

Summary

The manner in which the client software submits a request is described in Scenario SDSRV_Client_Request_Submission. The data server processing of the request is described in Scenario SDSRV_Server_Request_Handling. Once the request has been identified as a Search request, a GlParamterList is created to hold the set of commands which will be returned to the client-side of the system to update the client state to match the resulting DsSrWorkingCollection. A GlStatus is also created to store status information about the search. The DsSrCommand specifying the search, the GlStatus for passing status information and a GlParameterList for the results are provided to the DsSrWorkingCollection's ExecuteCommand method. This method in turn invokes the DsMdCatalog object's Search method. The processing of a science data query is described in Scenario SDSRV_Catalog_Search. The search returns a set of objects containing the metadata for the ESDT's which match the query constraints. The DsSrWorkingCollection iterates over the list of return objects, creating a DsGeESDTWrapper for each. The DsGeESDTWrapper associates a given set of metadata with the services available on that metadata. Each time a DsGeESDTWrapper object is created, the DsSrWorkingCollection puts an "add" directive in the GlParameterList. The manner in which the DsSrWorkingCollection returns the results to the DsSrRequest is described in Scenario SDSRV_Server_Request_Handling. The GlParameterList is returned as described in Scenario SDSRV_Asynchronous_Status_Updates.

Assumptions

- All corresponding client-side proxies have been established
- OODCE (or MSS) handles all of the telecommunications (finding objects, etc.)
- The WorkingCollection is empty at the start of this scenario
- Non-conflicting requests are handled asynchronously (i.e., queuing a request is an independent action from taking the next request from the queue

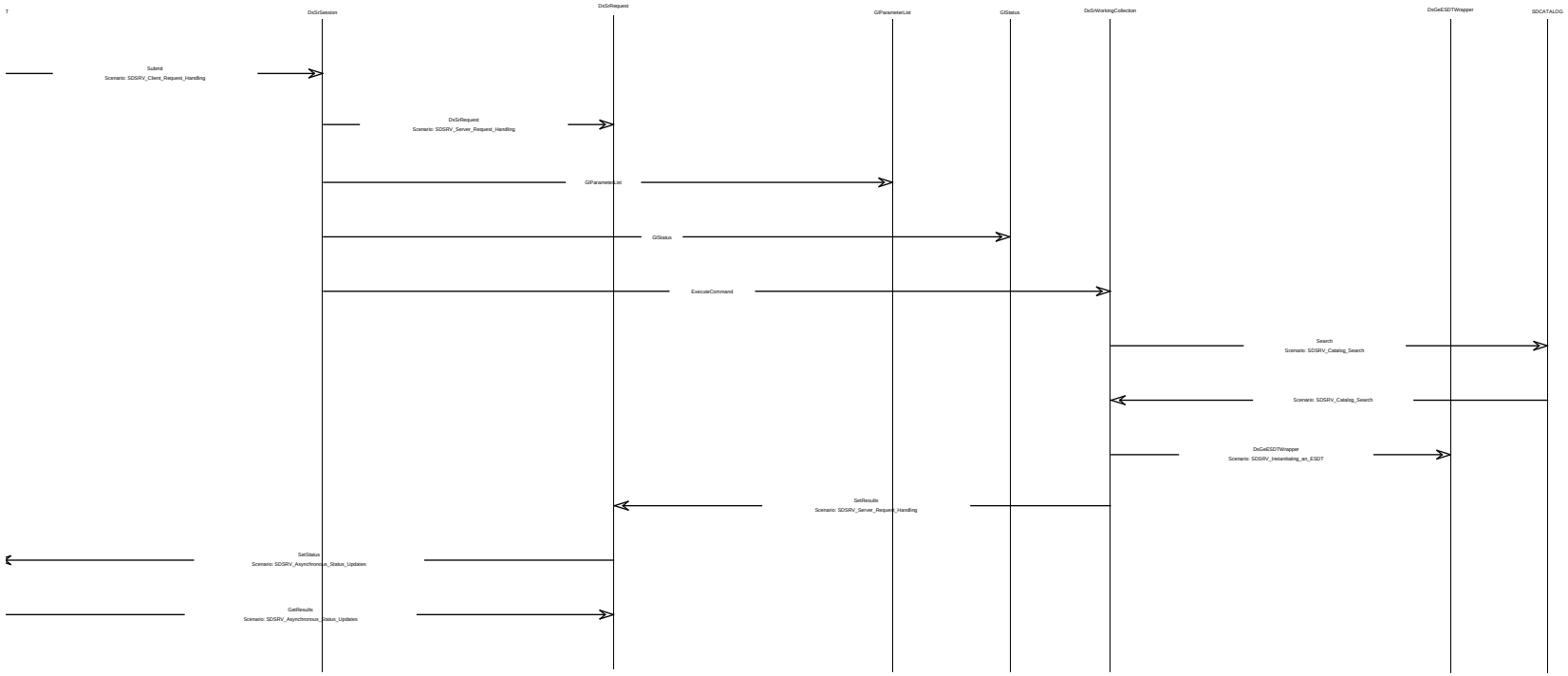


Figure 4.4-27. SDSRV_Server_Handling_a_Search_Request Dynamic Model

4.4.28 SDSRV_Server_Request_Handling

Summary

This scenario shows the interaction of classes in the data server to handle a newly submitted request. The DsClESDTReferenceCollector is a distributed object whose server-side is the DsSrSession. When the client submits the request to the collector, its core information is stored in the (distributed) DsClSubmittedRequest, and the unique ID of the DsClSubmittedRequest is sent to DsSrSession. DsSrSession uses this ID to find the DsClSubmittedRequest on the server side, and

extracts the information necessary to build a DsSrRequest. In this scenario, since it's the only request in the server, it's retrieved and executed immediately after it is queued by the DsSrSession. In reality, the execution of requests runs in a separate thread, and would be waiting for a request to arrive in the queue. As the request executes, each command is extracted from it, authorized against the client information that was provided when the DsSrSession was created, and executed. The scenario shows execution up to the point where the command is about to be handed to another class to be handled. This is where other scenarios will fill in specific command execution steps. Once the command has been executed, the server collects the results (see SDSRV_Client_Browsing) and cleans up.

Assumptions

- the new request is the only one in the server - the request contains exactly one command
- the client has all necessary permissions
- all steps execute successfully

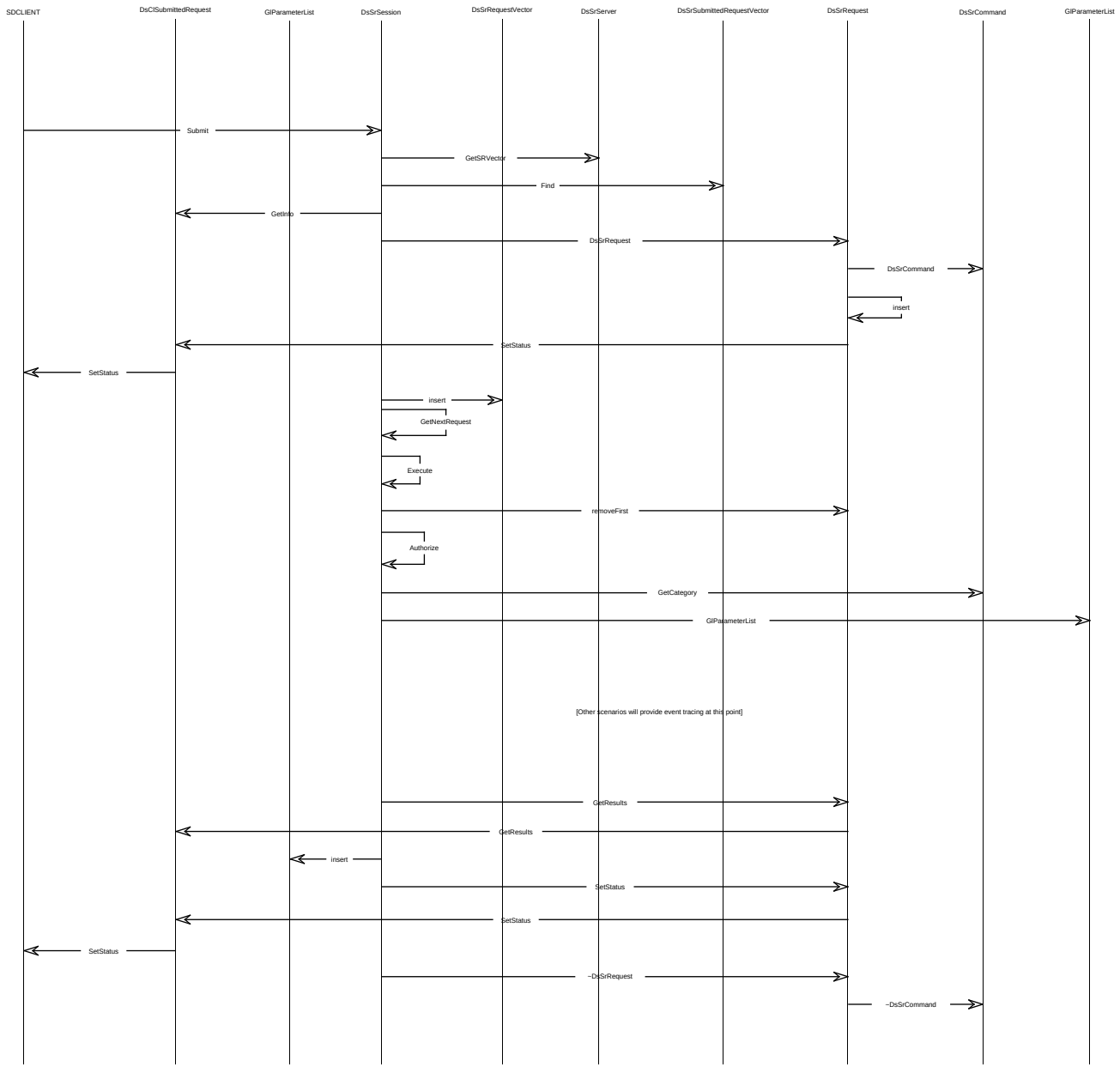


Figure 4.4-28. SDSRV_Server_Request_Handling Dynamic Model

4.4.29 SDSRV_Startup_of_a_Science_Data_Server

Summary

This scenario shows the steps required to startup a data server. The MSS_Object represents the management provided by the MSS subsystem over project entities such as processes. It will instantiate a DsCnDSSStartup object on the target system, which will handle all of the details of starting the data server. All server processes are started, which causes the creation of the indicated factory objects. Once the factories are created, new distributed objects may be created by clients to interface with the server. After starting the processes, DsCnDSSStartup instantiates a DsDeESDTDescriptorSet and initializes it, which walks through a list of all data types known for this server and creates a DsDeESDTDescriptor for each and initializes it. The result of initializing all of the types is the existence of advertisements for all of the services for every type, as well as advertisements for the events which are subscribable for each type. The creation of these advertisements can be traced in two other scenarios: SDSRV_Inserting_A_New_Data_Type and SDSRV_Registering_A_Subscribable_Event.

Assumptions

- all steps execute successfully

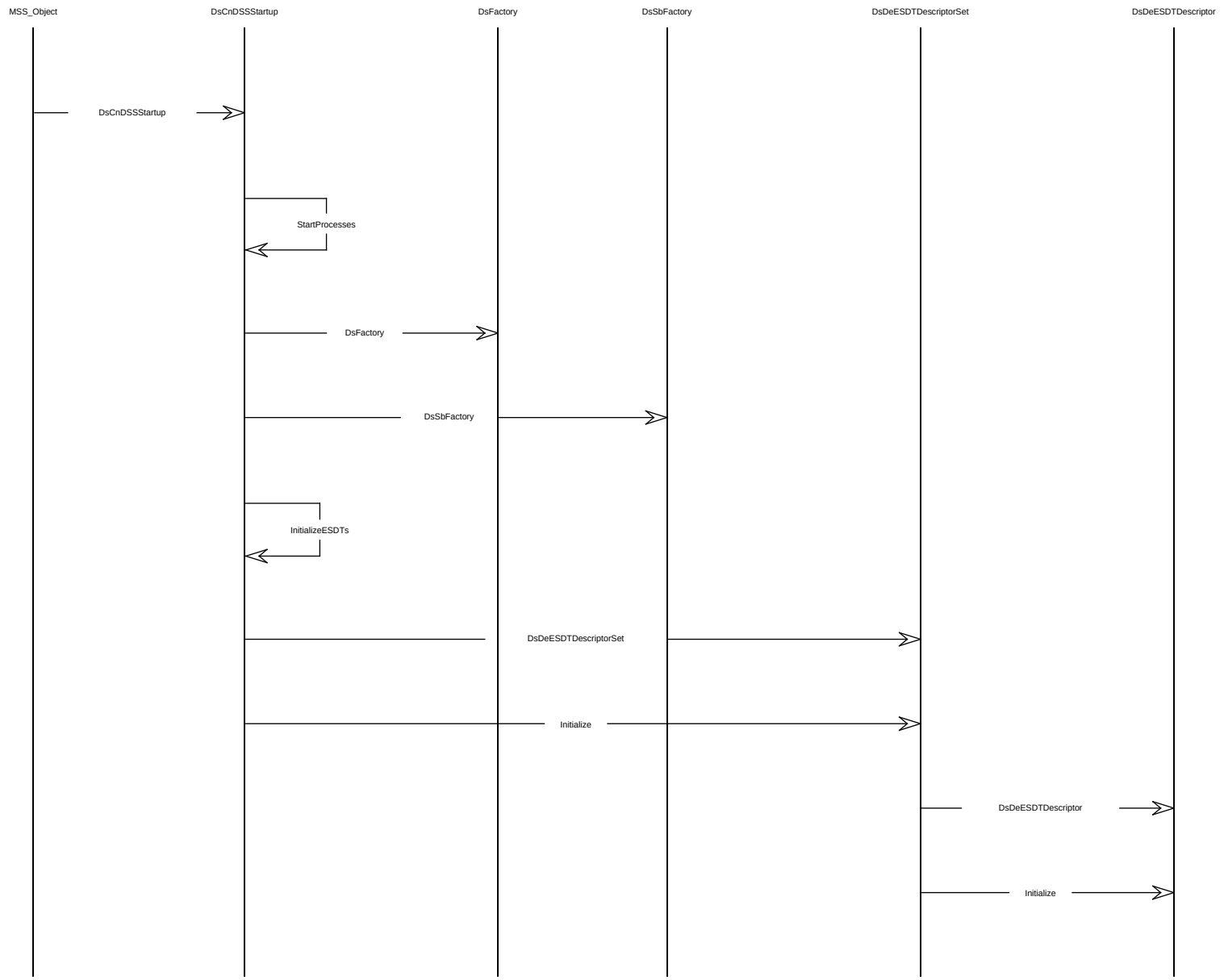


Figure 4.4-29. SDSRV_Startup_Of_A_Science_Data_Server Dynamic Model

4.4.30 SDSRV_Submitting_a_Subscription

Summary

This scenario shows how a client can submit a subscription to the SDSRV. In this scenario the client has retrieved a subscription advertisement from the Advertising server and has decided that he/she would like to be notified upon the future occurrence of that advertised event.

Assumptions

- Client has retrieved an advertisement object from the Advertising Server that represents the event that he/she is interested in.

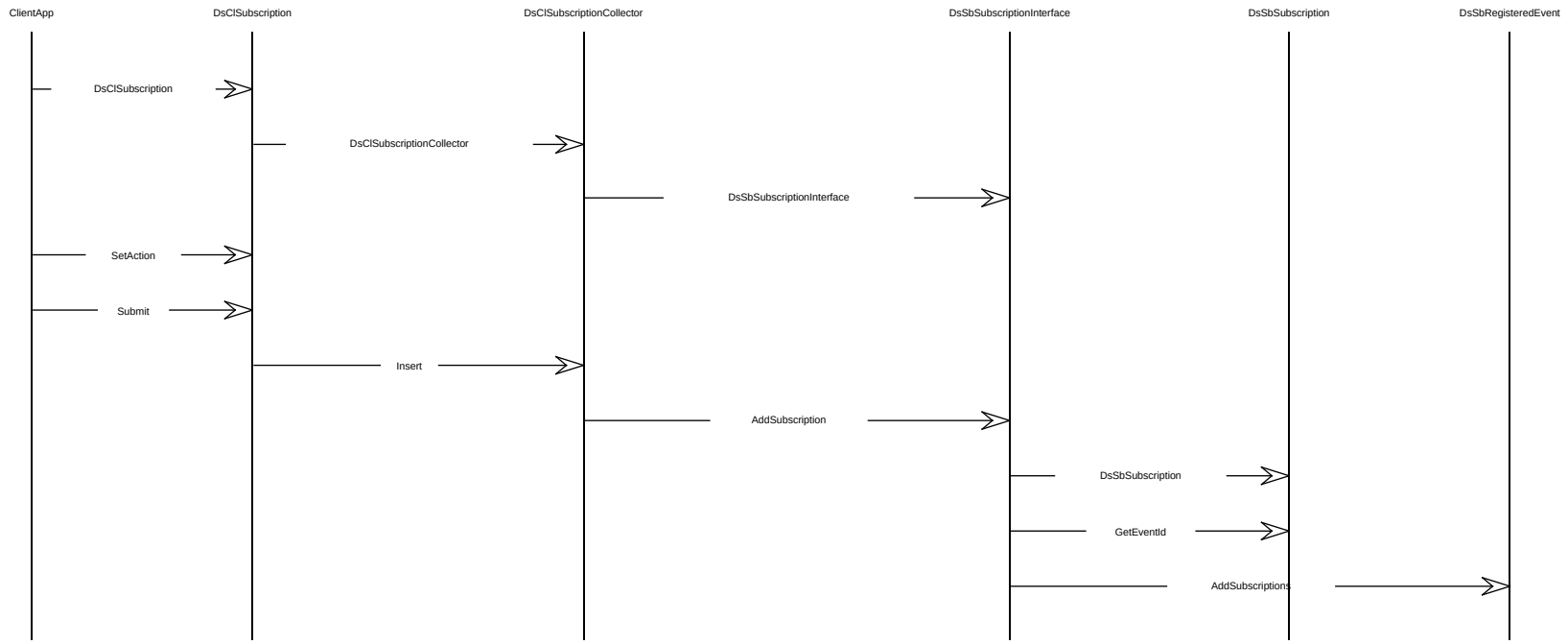


Figure 4.4-30. SDSRV_Submitting_a_Subscription Dynamic Model

4.4.31 SDSRV_Unregistering_a_Subscribable_Event

Summary

The purpose of this scenario is to show how a subscribable event which has been previously established, gets unregistered. This would be done if a Data Type were being removed from the Data Server. This scenario begins after an DsDeESDTDescriptor has been created and it has determined that it must unregister its subscribable events.

The DsDeESDTDescriptor establishes the DsGeESDTEventTable for this Data Type which contains the DsSbEvents that were previously stored when this Data Type was initialized. Because this is a distributed object, an DsSbEvent is created in the Subscription Server execution space. Then for each event that the Data Type has previously stored in the DsGeESDTEventTable, the DsDeESDTDescriptor unregisters that event. This causes the SubscriptionInterface to cancel the advertisement associated with this event. Finally, the DsSbEvent is removed from the persistent DsGeESDTEventTable.

Assumptions

- The SDSRV has the UR of its Subscription Server Factory.
- The Subscription Server is active.
- All events of a Data Type are being unregistered.

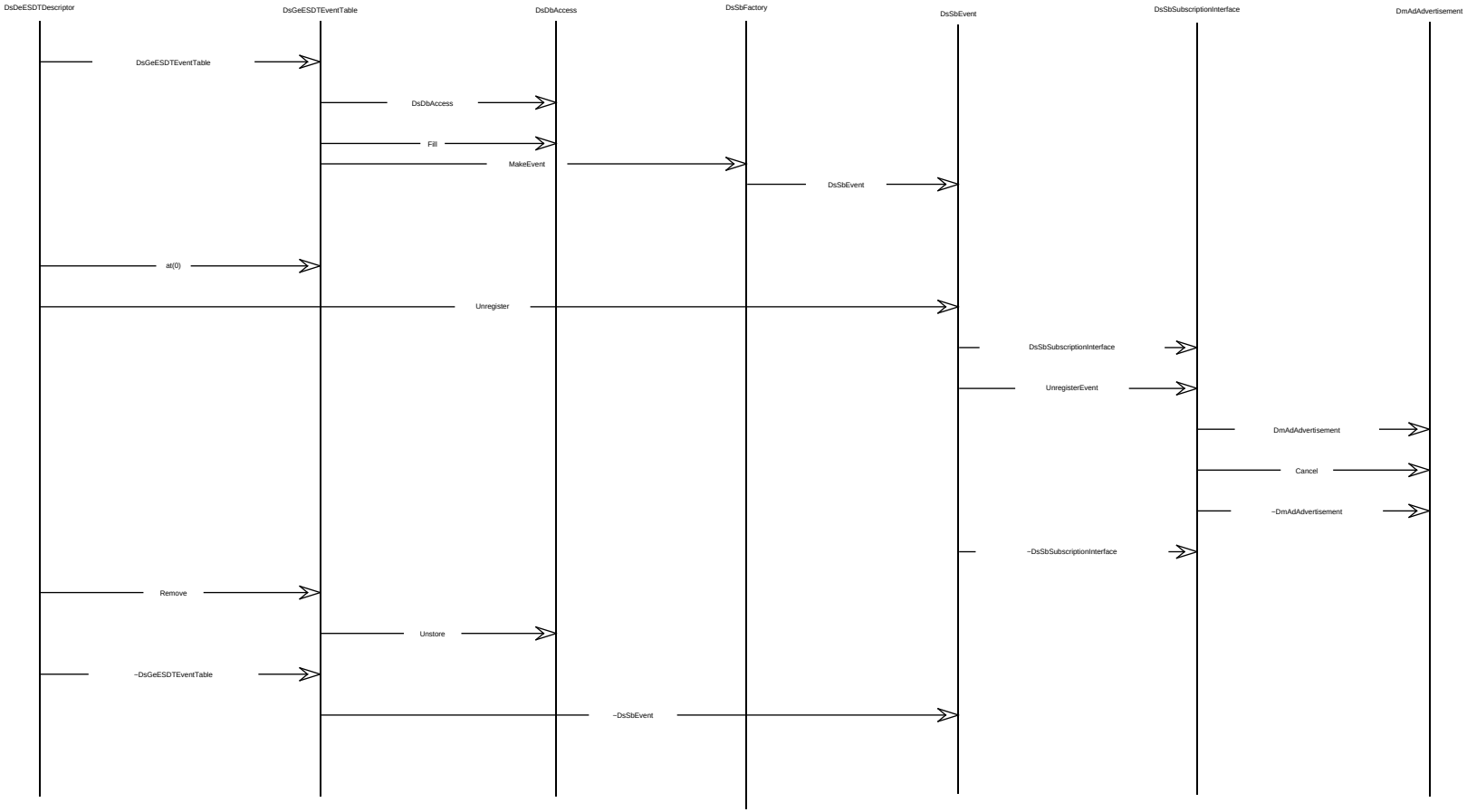


Figure 4.4-31. SDSRV_Unregistering_a_Subscribable_Event Dynamic Model

4.4.32 SDSRV_Update_Server_Configuration

Summary

This scenario shows how an operator may change the configuration of a SDSRV. In this scenario an operator has updated the threshold of queued requests for SDSRV connections in a GUI widget. The operator then selects an appropriate action widget (e.g. button) that triggers a callback, in this case the SetQueueSize method of DsAdRequestInterface object.

Assumptions

- The operator has already identified the SDSRV of interest
- Active connections that already have more queued request than the new threshold allows will continue processing, but no accept new requests until they have completed enough to get below the threshold.

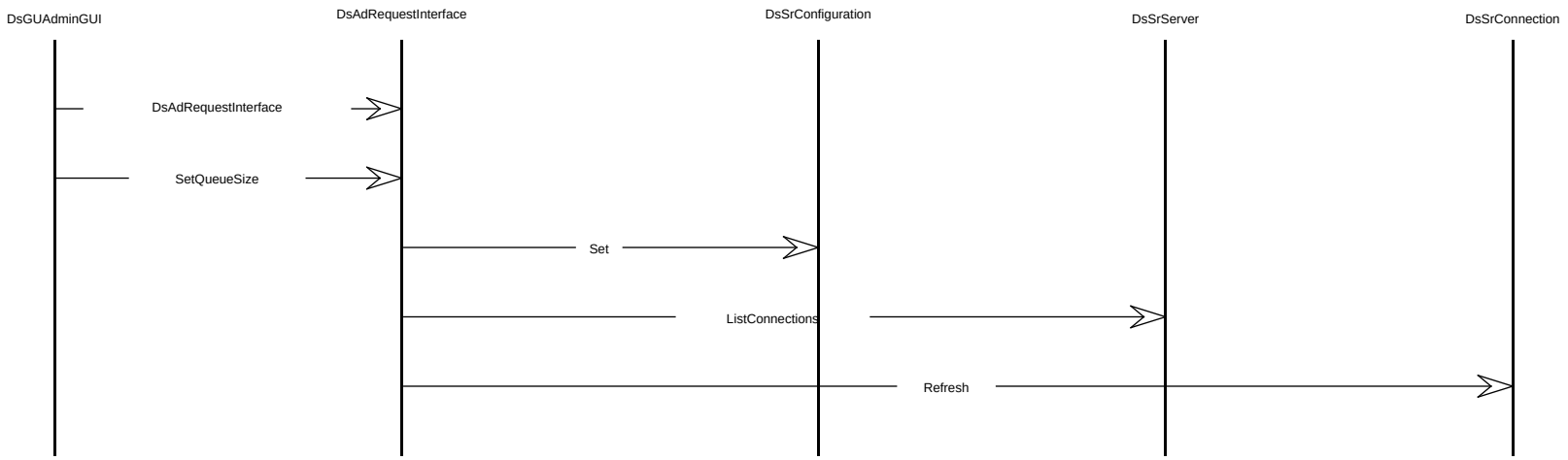


Figure 4.4-32. SDSRV_Update_Server_Configuration Dynamic Model

4.4.33 SDSRV_Updating_a_Subscription

Summary

The scenario shows how the client updates a subscription. The scenario shows steps used to update a subscription from the client side and ultimately, reflected on the server side. This is achieved through the distributed object, DsClSubscriptionCollector. The object, DsClSubscriptionCollector, has already been constructed. This scenario shows that the client wants to update the expiration date of the subscription. The client sets the expiration date of the subscription. The older subscription is withdrawn as outlined in the scenario SDSRV_Updating_a_Subscription. The subscription is submitted as outlined in the scenario SDSRV_Submitting_a_Subscription.

Assumptions

- user has logged on and has proper permissions.
- DsClSubscriptionCollector has already been created.

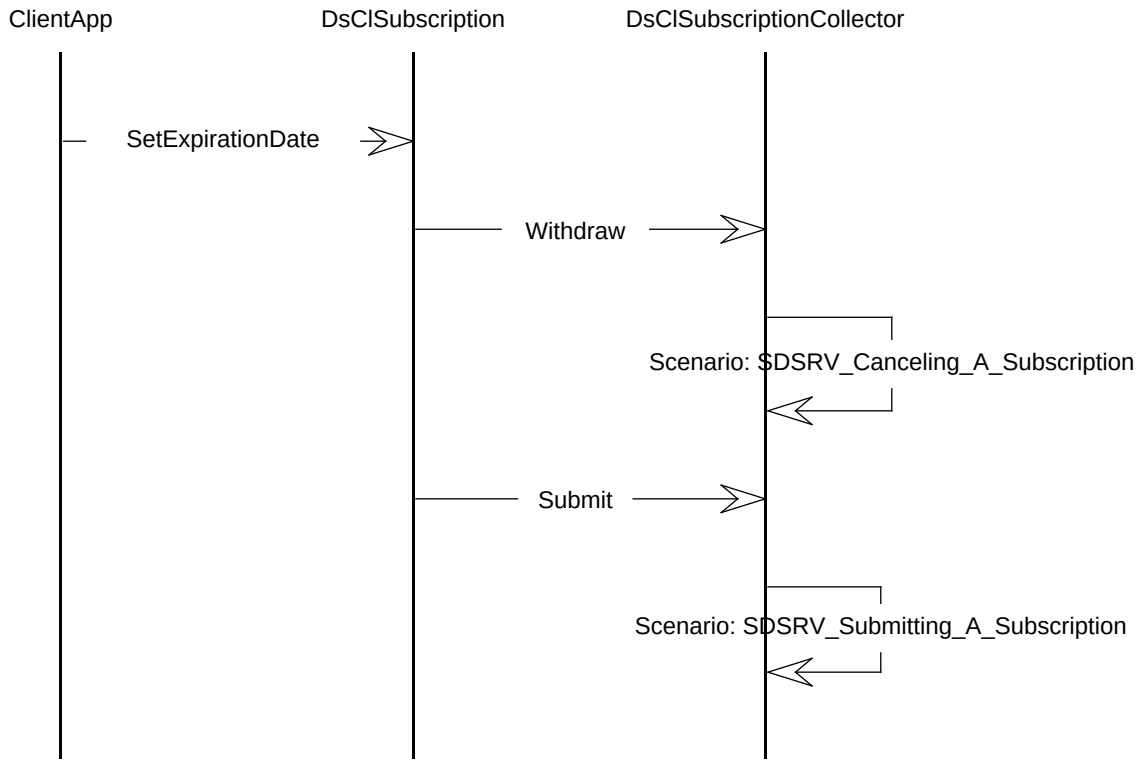


Figure 4.4-33. SDSRV_Updating_a_Subscription Dynamic Model

4.4.34 SDSRV_Validating_Metadata

Summary

The purpose of this scenario is to show how validation of metadata is accomplished within the Science Data Server. Metadata validation is accomplished by the Science Data Server for external clients such as the Ingest Subsystem and the Processing Subsystem. In addition, metadata validation occurs when a new granule is being inserted into the Science Data Server. This scenario assumes that the validation is being done in the context of an insertion of a new granule.

This scenario begins when an instance of an DsGeESDT creates the DsDeESDTDescriptor associated with its type and invokes the validation service. The client of this service, DsGeESDT in this case, must provide the P=V metadata to be validated and a GlStatus reference that will be updated during the validation process.

The first step in the validation process is to verify that the given metadata is valid with respect to the core metadata valid values. This DsDeCoreValidVector which is the same for all DsDeESDTDescriptors is created from its persistent location and it creates each of its DsDeCoreValid objects. The DsDeESDTDescriptor then looks at each entry in the given metadata and tries to find an associated entry in the vector. If an entry is found, the value of the input metadata entry is compared for validity against the vector entry. This is repeated for all metadata entries for the input metadata.

The second step in the validation process is to verify that the given metadata is valid with respect to metadata validation criteria that is specific to the Data Type. These valids are a refinement of the DsDeCoreValids. This process is nearly identical to the CoreValidVector validation except that the vector obtained from persistent storage in this case is specific to the given type.

The last step in the validation process is to ensure that all mandatory metadata entries are present in the given metadata. This is done with the MetadataDefVector that defines the metadata attributes for this data type. The MetadataDefVector is obtained from persistent storage and is used to look at each MetadataDef entry, see if it is mandatory, and ensure that the input metadata has included this attribute. An indication of the success or failure of the validation is provided as a result. If there are errors during the validation process, these will be recorded in the provided GlStatus object.

Assumptions

- The client of this service provided a P=V metadata stream.

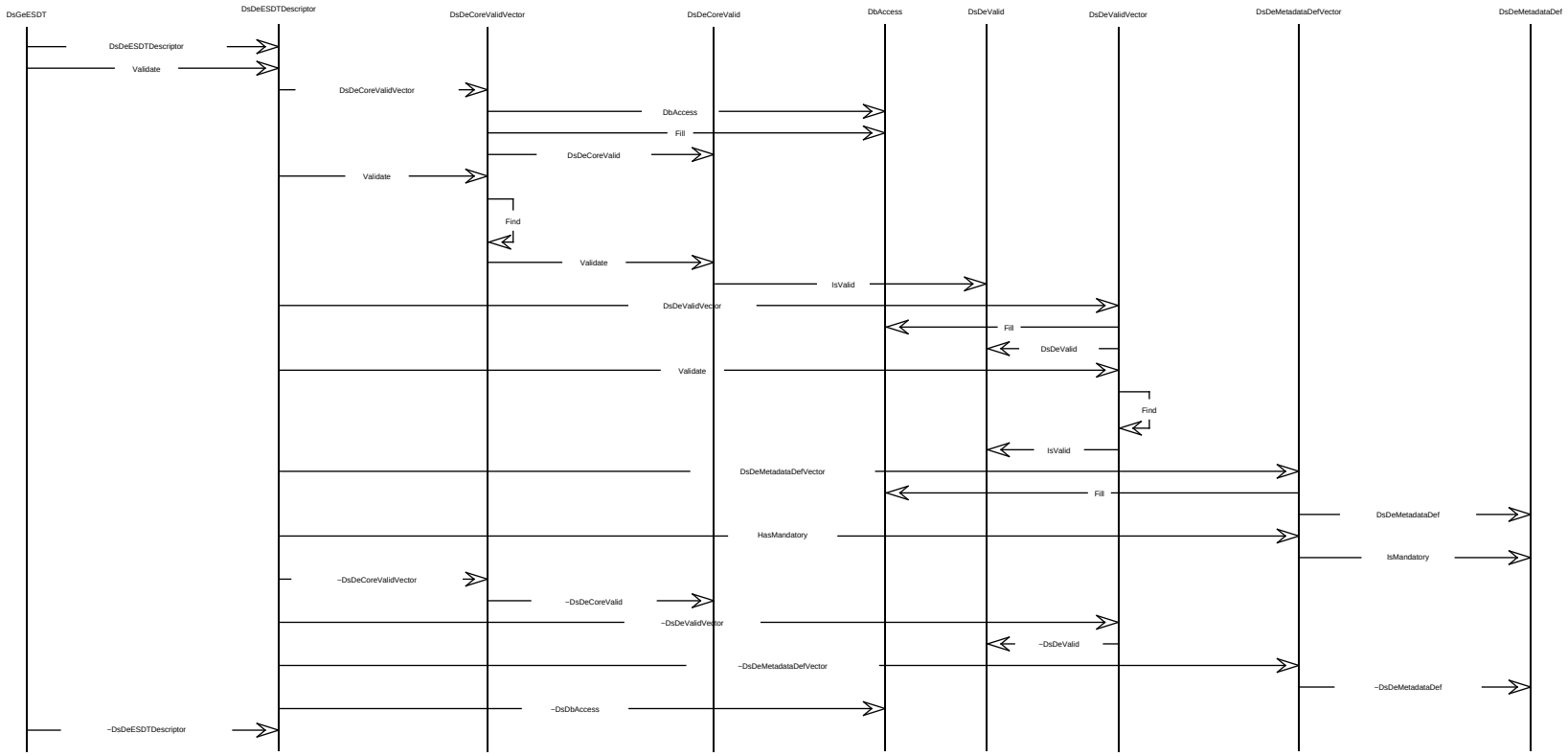


Figure 4.4-34. SDSRV_Validating_Metadata Dynamic Model

4.5 CSCI Structure

Table 4.5-1 shows the components of the SDSRV CI, (its CSC's). Each CSC is described and designated as being custom developed code (DEV), off-the-shelf (OTS) or a combination of the two (DEV/OTS). If the custom developed code will be used for OTS integration purposes, it is identified as WRAPPER. Following the table are a series of sections that describe each CSC and lists the classes that comprise that CSC.

Table 4.5-1. SDSRV's Components

CSC	Abbr.	Description	Implementation
Administration/Operation	Ad	The classes required to support Administrator and Operations functionality.	DEV
CERES	CE	The classes supporting the CERES Instrument Data Products.	DEV
Client	Cl	The Public classes offered to SDSRV clients. This includes the distributed classes.	DEV/OTS
Configuration/Startup	Cn	The classes supporting configuration and startup of the SDSRV.	DEV
Metadata	Md	The classes and database implementations supporting the storage and access of metadata.	DEV/WRAPPER
CSDT	CS	The CSDT classes.	WRAPPER
DB Wrappers	Db	The COTS database wrapper classes.	WRAPPER
Descriptors	De	The classes used to define and describe ESDTs and their supporting functionality.	DEV/OTS
General ESDT	Ge	Those classes which provide the framework around which specific ESDTs are built.	DEV
Global	Gl	Those classes that are intended to be globally available for re-use.	DEV/OTS
GUI	GU	The classes and other software that is supplied to support whatever GUI requirements that the SDSRV has.	DEV
LIS	LI	The classes supporting the LIS Instrument Data Products.	DEV
Non-Product Science ESDTs	NP	The classes used to provide the interface to and implementation of those ESDTs that are not ECS products, but whose content is still Earth Science data.	DEV
Non-Science ESDTs	NS	The classes used to support the ESDTs that do not contain Earth Science Data.	DEV
PR	Pr	The classes supporting the PR Data Products.	DEV
Server	Sr	The classes used to provide the framework for the infrastructure of the SDSRV.	DEV/OTS
Subscriptions	Sb	The classes used to support the subscription functionality.	DEV/OTS
TMI	Tm	The classes supporting the TMI Data Products.	DEV
VIRS	Vi	The classes supporting the VIRS Data Products.	DEV

4.5.1 CSC Definitions

4.5.1.1 Administration/Operation

Purpose and Description

The classes required to support SDSRVAdmininstrator and Operations functionality.

Classes

DsAdDataTypeCollector
DsAdDescriptor
DsAdLog
DsAdRequestInterface

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.2 CERES

Purpose and Description

The classes supporting the CERES Instrument Data Products.

Classes

DsCeCERES

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.3 Client

Purpose and Description

The Public classes offered to SDSRV clients. This includes distributed classes.

Classes

DsClAction
DsClCollector
DsClCollectorVector
DsClCommand
DsClConnection
DsClDescriptor
DsClDescriptorCollector
DsClESDTReference
DsClESDTReferenceCollector

DsClESDTReferenceVector
DsClNotificationReceiver
DsClQuery
DsClRequest
DsClRequestVector
DsClSubmittedRequest
DsClSubscription
DsClSubscriptionCollector
DsClTypeInfo

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.4 Configuration/Startup

Purpose and Description

The classes supporting configuration and startup of the SDSRV.

Classes

DsCnConfiguration
DsCnDSSConfiguration
DsCnDSSStartup

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.5 Metadata

Purpose and Description

The classes and database implementations supporting the storage and access of metadata.

Classes

DsMdCatalog
DsMdMetadata

Candidate Products

Sybase/SQS

ECS White Paper References

Not Applicable

4.5.1.6 CSDT

Purpose and Description

The CSDT classes.

Classes

DsCsCSDT
DsCsGrid
DsCsImage
DsCsPoint
DsCsRaw
DsCsSwath

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.7 DB Wrappers

Purpose and Description

The COTS database wrapper classes.

Classes

DsDbAccess
DsDbAttributeToTableVector
DsDbConnection
DsDbGranuleToDbVector
DsDbInterface

Candidate Products

Sybase

ECS White Paper References

Not Applicable

4.5.1.8 Descriptors

Purpose and Description

The classes used to define and describe ESDTs and their supporting functionality.

Classes

DsDeCoreValid
DsDeCoreValidVector
DsDeDD
DsDeDDVector
DsDeESDTDescriptor
DsDeESDTDescriptorSet

DsDeEvent
DsDeEventVector
DsDeMathOp
DsDeMetadataDef
DsDeMetadataDefVector
DsDeRange
DsDeScienceParameter
DsDeScienceParameterVector
DsDeSeries
DsDeService
DsDeServiceVector
DsDeStaticMetadata
DsDeStaticMetadataVector
DsDeValid
DsDeValidVector

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.9 General ESDT

Purpose and Description

Those classes which provide the framework around which specific ESDTs are built.

Classes

DsGeBrowseProduct
DsGeDynamicLibrary
DsGeECSDataProduct
DsGeESDT
DsGeESDTConfiguration
DsGeESDTDynamicLibrary
DsGeESDTEventTable
DsGeESDTServiceProvider
DsGeESDTWrapper
DsGeRange
DsGeScienceData
DsGeSummaryProduct
DsGeTypeID

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.10 Global

Purpose and Description

Those classes that are intended to be globally available for re-use.

Classes

GlBinaryP
GlDateP
GlDoubleP
GlLongP
GlParameter
GlParameterList
GlStringP
GlTimeP

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.11 GUI

Purpose and Description

The classes and other software that is supplied to support whatever GUI requirements that the SDSRV has.

Classes

DsGuAdminGUI

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.12 LIS

Purpose and Description

The classes supporting the LIS Instrument Data Products.

Classes

DsLiLIS

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.13 Non-Product Science ESDTs

Purpose and Description

The classes used to provide the interface to and implementation of those ESDTs that are not ECS products, but whose content is still Earth Science data.

Classes

DsCoCombination
DsGvRadar
DsNmNMC
DsNpAncillary
DsNpCalibration
DsNpCorrelative
DsNpNonECSDataProduct
DsNpOA
DsNpPlatform
DsNpVersion0
DsSsSSMI

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.14 Non-Science ESDTs

Purpose and Description

The classes used to support the ESDTs that do not contain Earth Science Data.

Classes

DsNsHistoricalData
DsNsMPR
DsNsNonECSDataProduct
DsNsProductionHistory
DsNsQAStatistics
DsNsScienceSoftwareArchivePackage

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.15 PR

Purpose and Description

The classes supporting the PR Data Products.

Classes

DsPrRadar

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.16 Server

Purpose and Description

The classes used to provide the framework for the infrastructure of the SDSRV.

Classes

DsFactory
DsSrClient
DsSrCommand
DsSrCommandBase
DsSrCommandInfo
DsSrCommandVector
DsSrConnection
DsSrQueuedConnection
DsSrRequest
DsSrRequestBase
DsSrRequestInfo
DsSrRequestVector
DsSrServer
DsSrSession
DsSrSubmittedRequestVector
DsSrWorkingCollection

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.17 Subscriptions

Purpose and Description

The classes used to support the subscription functionality.

Classes

DsSbAction
DsSbActionBase
DsSbCallBackTimer
DsSbEvent

DsSbEventHandler
DsSbEventTimer
DsSbRegisteredEvent
DsSbSubscription
DsSbSubscriptionInterface
DsSbTimer

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.18 TMI

Purpose and Description

The classes supporting the TMI Data Products.

Classes

DsTmTMI

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.19 VIRS

Purpose and Description

The classes supporting the VIRS Data Products.

Classes

DsViVIRS

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.2 CSCI Dynamic Architecture

The dynamic architecture of the SDSRV CI is defined by the individual processes (or executables) that comprise its implementation. The SDSRV CI is implemented as three separate, cooperating UNIX processes. Those processes are the ScienceDataServer, SDSRVAdmin and SubscriptionServer. These three processes, along with those executables that include the SDSRV Client CSC library, will supply the SDSRV CI functionality to ECS. The implementation of the SDSRV CI also depends on the various implementations of two of the other DSS software CIs

(DDIST and STMGT CIs). The following paragraphs describe each of the three SDSRV CI processes, along with some general information about other applications that will use the SDSRV's Client CSC.

4.5.2.1 ScienceDataServer Process

The primary role of the ScienceDataServer Process is to provide an access mechanism for all the advertised data type services and their implementations. For systems external to ECS and for ECS subsystems outside the DSS, the ScienceDataServer Process is the repository of all ECS non-document data objects and the interface to their services.

This process depends on the SubscriptionServer Process as well as the implementation process from the STMGT and DDIST CI's of the DSS. It also depends on implementations of various CSS and MSS services.

The ScienceDataServer Process comprises the following SDSRV CI CSCs:

- Configuration/Startup
- Metadata
- CSDT
- DB Wrappers
- Descriptors
- General ESDT
- Global
- Non-ProductScienceESDTs
- Non-ScienceESDTs
- Server
- Subscription

It also comprises the interface class CSCs from STMGT, DDIST, ADSRV, MSS and CSS. Additionally, the Release A implementation of the ScienceDataServer Process will comprise implementations for the Release A data types. These implementations are the CERES, LIS, PR, TMI and VIRS SDSRV CSCs.

4.5.2.2 SDSRVAdmin Process

The role of the SDSRVAdmin Process is to support Science Data Server Administrators and Operators in performing their role in ECS. SDSRV administrators and operators for Release A are primarily concerned with the configuration of the SDSRV and monitoring and controlling the processing within the ScienceDataServer Process.

This process depends on the ScienceDataServer Process as well as the implementations of various CSS and MSS services.

The SDSRVAdmin Process comprises the following SDSRV CI CSCs:

- Administration/Operation
- Client

- Configuration/Startup
- DB Wrappers
- Global
- GUI
- Server

It also comprises the interface class CSCs from MSS and CSS.

4.5.2.3 SubscriptionServer Process

The role of the SubscriptionServer Process is to support the detection of previously defined events and to perform specified actions on behalf of clients who have previously registered to those events.

The SubscriptionServer Process depends on the implementations of various ADSRV, CSS and MSS services. In order to actually perform its role in the ECS, the SubscriptionServer Process requires that its client, namely the ScienceDataServer Process, be active and have previously registered events with it.

The SubscriptionServer Process comprises the following SDSRV CI CSCs:

- Client
- Configuration/Startup
- Global
- Subscription

It is also comprised of the interface class CSCs from ADSRV, CSS and MSS.

4.5.2.4 General Application of SDSRV Client CSC

The applications that utilize ScienceDataServer Process capabilities are generically referred to as "SDSRV Clients Applications", or "Clients" within the SDSRV context. There are a number of ECS SDSRV Clients Applications as well as, potentially, non-ECS SDSRV Clients. Examples of ECS Clients are the Ingest, Planning, Processing and CIDM subsystems. All of these Clients must use the SDSRV Client CSC implementation. The implementation of this CSC is in the form of a UNIX library. Public interfaces to the Client CSC will be provided in the form of header files that may be included in the Client's source code. The Client process must be linked with the SDSRV Client CSC implementation library. The public interface, as well as the implementation, of the SDSRV Client CSC will be constructed in a way that will allow SDSRV Client Applications to extend the Client CSC to more closely fit their own needs through the C++ implementation of inheritance. The Client CSC's role is to provide an extensible interface to, and abstraction of the ScienceDataServer Process. This CSC will hide from the Client the details of network location, OODCE and other technologies used in support of the SDSRV CI's implementation. Restrictions on SDSRV Client Applications include those imposed by the technology chosen for distributed communication in ECS, namely OODCE. Indeed, applications that include the SDSRV Client CSC implementation are, by definition, OODCE client applications. OODCE restrictions include the requirement that SDSRV Client Applications exist within a DCE cell.

4.6 SDSRV CSCI Management and Operation

The materials in the following paragraphs discuss the management and operations of software components discussed in section 4.5.

4.6.1 System Management Strategy

The SDSRV CSCI is designed to provide robust data storage, search, and distribution services to external data providers and requestors. Specifically, the design goal of the SDSRV CSCI is to always return status (successful or unsuccessful) for every received request. To accomplish that goal, the CSCI follows ECS project guidelines for:

- Process startup and shutdown;
- Error detection and reporting;
- Fault tolerance and error recovery

4.6.1.1 Startup/Shutdown

MSS provides life-cycle services for system startup and shutdown. The ScienceDataServer and SubscriptionServer SDSRV processes act as an "object factory". As such each process instantiates objects in process threads (pthreads) when a request is serviced. At DSS system startup, three required processes are launched. The ScienceDataServer process is started as a standalone processes. The primary role of the ScienceDataServer Process is to provide an access mechanism for all the advertised data type services and their implementations. The SubscriptionServer process will also routinely be started at system startup. The role of the SubscriptionServer Process is to support the detection of previously defined events and to perform specified actions on behalf of clients who have previously registered to those events. The third "process" that is required for SDSRV functionality is the COTS DBMS product. The DBMS will be used for persistent storage of science and operational system data.

The final process associated with the SDSRV is the SDSRVAdmin process. The role of the SDSRVAdmin Process is to support Science Data Server Administrators and Operators in performing their role in ECS. SDSRV administrators and operators for Release A are primarily concerned with the configuration of the SDSRV and monitoring and controlling the processing within the ScienceDataServer Process. The SDSRVAdmin process will be routinely started and used for control and operation, but technically is not required for unattended operations of the SDSRV CSCI.

4.6.1.2 Error Detection and Reporting

The Data Server CSCI is designed for primarily automated operations with little need for operations involvement short of tuning and critical error conditions. CSS and MSS jointly provide event logging services for logging and reporting errors and faults, for browsing error/status logs, and for detecting and reporting critical errors. The Data Server CSCI will fully use those services during operations. Errors/status may be reported in two error logs. MSS maintains the first log, the MSS event log. It contains errors/status of interest to operations staff to evaluate system status and to perform trend analysis. The Data Server subsystem maintains the second log. The Data

Server event log contains selected errors/status from the MSS event log (for context) plus highly-detailed debug events. Software maintenance personnel use the Data Server event log to diagnose system and software problems in response to trouble tickets.

Non-critical errors encountered during processing that will be handled at the application level will be fully resolved and enumerated during development. Major conditions that require operator intervention and/or are considered catastrophic in the processing of requests are listed in Table 4.6-1.

Table 4.6-1. SDSRV Error Categories

Error Category	Actions to Be Taken
Initialization File/Environment Corrupt	This would be seen during a system startup process and would result in one or more executables not starting. Operations staff evaluate the condition and correct.
DB Transaction Log Full	This condition would result in the inability to insert additional persistent data into the Data Base engine(s) used within Data Server. Operations (DBA) would need to dump the transaction log and restore proper operation of the affected Data Base.
DB Connection Dropped	This could be a serious failure of the Data Base or a short lived problem with the connection. Operations (DBA) would need to evaluate the problem, possibly restarting the Data Base and the Data Server processes.
Unable to establish link to/invoke DLL	The use of DLLs is a fundamental aspect in the Data Server CSCI design. DLLs will be installed and maintained in the system for established and newly added data type services. The inability to utilize or invoke any of these libraries would be a fundamental error in the configuration and operation of the system. Operations would issue a trouble ticket and need to have the problem analyzed and resolved.
Internal queue overflow	Errors reported along these lines represent a very poorly tuned and/ or faulty system. This type error would represent potential loss of service requests. Operations staff would immediately throttle back system processing thresholds and write a trouble ticket for future off line analysis and tuning.
Unable to allocate disk space	Unable to allocate working storage space using Data Server STMGT CSCI services. This is another system of a poorly tuned system. Report alert to operations staff who would immediately lower system thresholds for requests. Operations staff would analyze system off-line and tune.
dbcc error detection	The Sybase Database Consistency Checker (dbcc) utility will be run periodically to check for errors in database allocations for databases, catalogs, tables and indexes. When an error is reported the operator or Database Administrator will run the dbcc utility with the "fix" option to correct the problem.

4.6.1.3 Fault Tolerance and Error Recovery

Once a service request is accepted from a client (client being defined as any service requestor, not just the ECS Desktop Client), it is the Data Server Subsystem design goal to complete the request processing and return status (successful or unsuccessful) to the requestor. The Data Server CSCI is built on the model of checkpointing processing at the command level (within a request) along with the user's working collection (context in a manner of speaking). During restart or recovery operations the CSCI will restore a user's working collection and resume processing at the next unprocessed command. This model actually evolves over time into the suspend/resume feature of the CSCI (Release B). Therefore, upon establishment of a user session, the SDSRV CSCI will have the user's working collection checkpointed to the COTS data base. Likewise, Requests (containing Commands) are also checkpointed. After each command is completed, the working collection will be checkpointed if it has changed in value or state. After a process or system failure, the checkpointed working collection and Requests are automatically restored to the last checkpointed state and processing continues.

Failure scenarios with recovery methods:

a. Failure of a Data Server Executable. This process is immediately restarted as a Unix standalone process. The User Working Collections and Requests are restored from their checkpointed states. Data Base integrity is verified and transactions rolled back via COTS procedures.

b. Loss of the data base tables used for checkpointing. The data base management system automatically logs transactions to allow restorations of table information. This feature, coupled with DBA generated Data Base backups, provide for recovery. Since high reliability is required in this area, the Data Base tables will be stored on RAID.

c. Failure of the processor on which a SDSRV process is running. In general, the processor automatically restarts. Restart of individual processes is handled as a combination of one or more of the above process restarts. If the processor is disabled, the disablement is detected my MSS SNMP services and a backup processor is restarted. The backup processor has full access to the data base tables used for checkpointing. Again, restart of individual processes is handled as described above.

d. Failure of an external application. After a given number of retries to transmit data or via DCE services that alert us of failure of the recipient process, operations staff are notified by means of an alert message. The DAAC operations staff will coordinate to diagnose the failure.

4.6.2 Operator Interfaces

DAAC operations personnel are provided with an X-Windows/Motif-based GUI to access operations data bases and system configuration information. Table 4.6-2 highlights the critical SDSRV GUI screens.

Table 4.6-2. Science Data Server Subsystem Administration Management GUI

GUI	Description	Data	Operations
Science Data Server Administration GUI	Primary Screen for Data Server Requests.	References to: Request, Log&Reports, Configuration, Client Management and Resource Management GUIs.	Logon Realizes and/or makes the GUIs visible.
Data Type Management GUI	Allows operators to manage the data types offered by the Science Data Server	List of data types (ESDTs) Descriptor contents Related Libraries	View Data Types View a specific Data Type Update a descriptor Add a Data Type Remove a Data Type
Subscription Management GUI	Allows operators to manage the current subscriptions. Creation, update, deletion of subscriptions will be managed using the standard Client Subscription GUI	Event Information: Event ID Event Description Subscriptions on event Subscription Information: Event ID Client Information Subscription Type Subscription Action	View Events View Subscriptions Filter Subscriptions (event, client, action) Access to Client Subscription GUI
Database COTS GUI	COTS provided GUI to support database management	COTS	COTS

4.6.3 Standard SDSRV Reports

Table 4.6.3-1. Standard Science Data Server Reports

Report Type	Report Description	Intended Audience
User Characterization	Is a summary report identifying each user class and the number of users in each class. The reporting period is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date. The report includes: the number of new distinct users per time period, the number of repeat distinct users per time period, summary of users by class, by product interest, by mode of access, by user affiliation.	System Operator Resource Planner Performance Analyst Sr. Science Coordinator Operations Supervisor DAAC Manager
System Access Profile	A summary report providing the profile of user system accesses and the time spent by each user on the system. The reporting period is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date. Information provided includes: Number of Search/Browse requests for the time period, the number of returned browse items for the period, the number of separate system accesses by DAAC staff, the number of separate system accesses by non-DAAC staff including guest users and captive accounts, total number of separate accesses, total average session time over all user sessions, total number of distinct users, total number of guest users, total users accesses identified by access method (e.g. ftp).	System Operator Resource Planner Performance Analyst Sr. Science Coordinator Operations Supervisor DAAC Manager
Science Data Server Error	The Science Data Server Error Report is a summary report of the frequency of errors of different types encountered during server operations. The report consists of an Error Class Summary which counts the reported errors for each error class and type.	System Operator Performance Analyst Operations Supervisor DAAC Manager

This page intentionally left blank.

5. DDSRV - Document Data Server CSCI

5.1 CSCI Overview

The Document Data Server (DDSVR) reflects many of the design principles used by the SDSVR in its management of Earth Science Data Types. The DDSVR will have implementation differences due to requirements unique to document search and retrieval, and the constraints placed by WWW technology.

The design of the CI has been driven by the commitment to implementing the "data are data" vision, the desire to retain design symmetry with the Science Data Server (SDSVR), the requirement to interface with the V0/V1 Gateway services, the desire to minimize V0 document migration costs, and the mandate to maximize the use of COTS, especially, WWW technology, in the management of Earth Science document data types.

With these design goals in mind, the DDSVR maximizes the reuse of common functionality with the SDSVR. Common base classes are used for client, server, command and request objects. This provides a common interface for constructing and executing service requests. Similarly, common data modeling abstractions are employed. Documents are modeled within a specialization hierarchy derived from the ESDT base class. ESDTs are implemented using a CSDT specialization hierarchy representing different document storage formats.

Services are statically bound to document objects. A stateless communication paradigm is used reducing the design complexity. Object association multiplicity has been minimized within the design, reflecting the design drivers behind the DDSVR.

A key design driver of the Data Server is the maintenance of a common data model for ECS data objects. Although distinct programmatic interfaces will be implemented for the DDSVR and SDSVR, a single common data model will be internally managed in order to support the navigation between science and document objects, while maintaining integrity between the data objects in an automated fashion. This approach is anticipated to reduce evolution costs when a common set of data type services will be used to access both science and document objects.

5.2 CSCI Context

The external interface to the DDSVR is through HTTP. Search and acquire requests are formulated as WAIS queries and HTTP Get commands respectively and issued by the V0 Client Guide Subsystem using HTTP. For those document types not currently accessible through the V0 Client, a Forms interface on a WWW Client will be used to formulate identical requests. These requests are received directly by the DDSVR. A HTTP demon process monitors a TCP/IP port for incoming requests and hands them off to the DDSVR using a CGI call. These requests are packaged as internal command and request structures by a client object which is then executed by the server object.

Documents are ingested into the ECS via the Ingest Subsystem. Insert requests are constructed by Ingest specifying the document type, file location and metadata which is then submitted to the DDSVR. The metadata is inserted into the DBMS through the DBMS Wrapper layer, the

document indexed for free text queries using COTS indexing technology and the document data inserted into the document repository.

Document search requests are received in WAIS query format which are interpreted into an internal query format and submitted to the DBMS Wrapper. The search results are formatted in HTML and returned across the HTTP connection to be rendered in the WWW Client.

Document acquire requests are received as HTTP Get commands which are interpreted as extract commands to access documents in the document repository. A HTTP header is constructed specifying the MIME type of the document and the document data returned across the HTTP connection to be handled by the WWW Client.

Figure 5.2-1 provides a context diagram for this CSCI which shows the interfaces with other CSCIs.

Table 5.2-1 provides a functional grouping of these interfaces along with the associated input and output data.

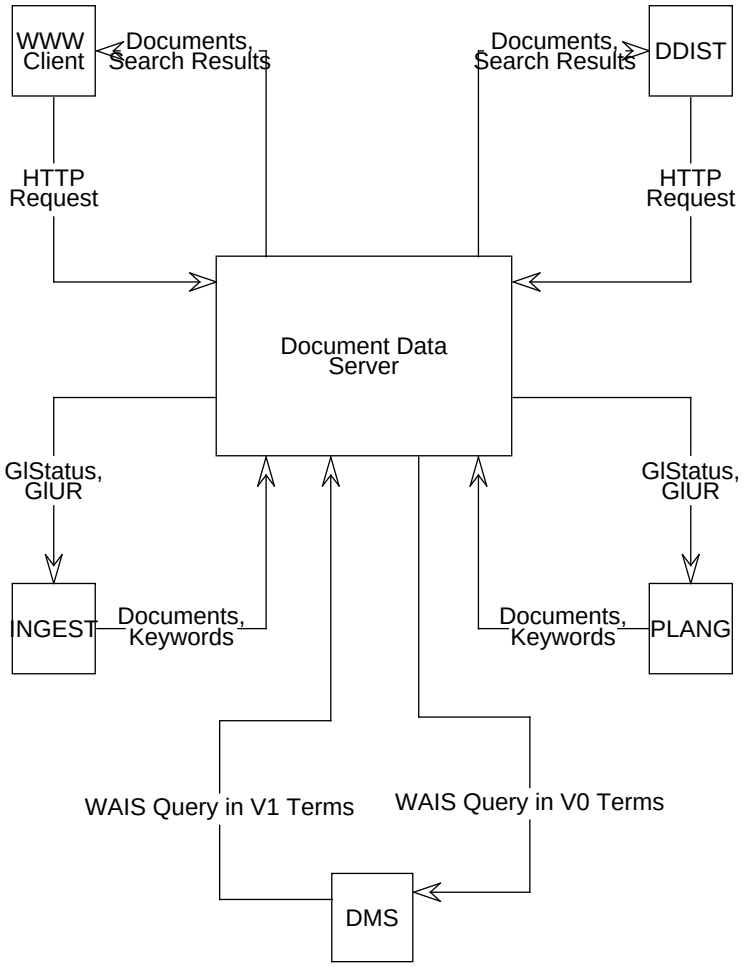


Figure 5.2-1. DDSRVContext Diagram

5.2-1. DDSRVContextevents Event Flow Summary Table

Sender	Receiver	Event Name	Detailed Signature
Document Data Server	WWW Client	Documents	
INGEST	DocumentDataServer	Documents	
PLANG	DocumentDataServer	Documents	
DocumentDataServer	DDIST	Documents	
DocumentDataServer	PLANG	GlStatus	
DocumentDataServer	INGEST	GlStatus	
DocumentDataServer	PLANG	GlUR	
DocumentDataServer	INGEST	GlUR	
WWWClient	DocumentDataServer	HTTP Request	
DDIST	DocumentDataServer	HTTP Request	
PLANG	DocumentDataServer	Keywords	
INGEST	DocumentDataServer	Keywords	
DocumentDataServer	DDIST	Search Results	
DocumentDataServer	WWWClient	Search Results	
DocumentDataServer	DMS	WAIS Query in V0 Terms	
DMS	DocumentDataServer	WAIS Query in V1 Terms	

5.3 CSCI Object Model

This section provides an object model for the Document Data Server CSCI. The Document Data Server Object model contains two main categories of objects. The first are the generic server objects that support access to the Document Data Server. These objects include the Server, Client, Request, and Command. The second category of objects are the Earth Science Data Type (ESDT) objects and supporting objects. This includes the Guide, Algorithm Description, Production Plan and Reference Paper. The supporting objects for the ESDTs include the Computer Science Data Types (CSDT) that implement the ESDTs (ASCII, HTML, PDF, RTF, Postscript), and those required for exporting configuration information to other CSCIs (Keyword, Keyword Locator). Other objects appear in the object model, but are not described in the following section; they have been allocated to other CSCIs and are replicated here for purpose of completeness.

General Notes: In the following object class descriptions, classes always inherit attributes, operations, and associations from their parent classes. Where all attributes in the current design are inherited from the parent class, this is indicated by the text "*All Attributes inherited from the parent class*" (and analogously for operations). If a derived class has additional attributes of its own, those new attributes are listed, but the attributes from the parent class are not repeated (analogously for operations). "*None"* may indicate that the preliminary design has not resulted in any specific attributes or operations (though over the course of the detailed design, such items may be added).

The Object Classes in the Object Model diagrams and descriptions each have an identified set of attributes and public operations. The listed public services do not include two operations: a constructor (often referred to as "Create") and a destructor (often referred to as "Destroy"). These operations are not listed, but implied for all Object Classes in these models. In order to implement these Object Classes constructors and destructors must be included, hence they are implied at the preliminary design level. The specific constructor and destructor operations will be identified through the detailed design activities. During the detailed design phase any overloading of constructors and destructors (as well as other services) will also be defined.

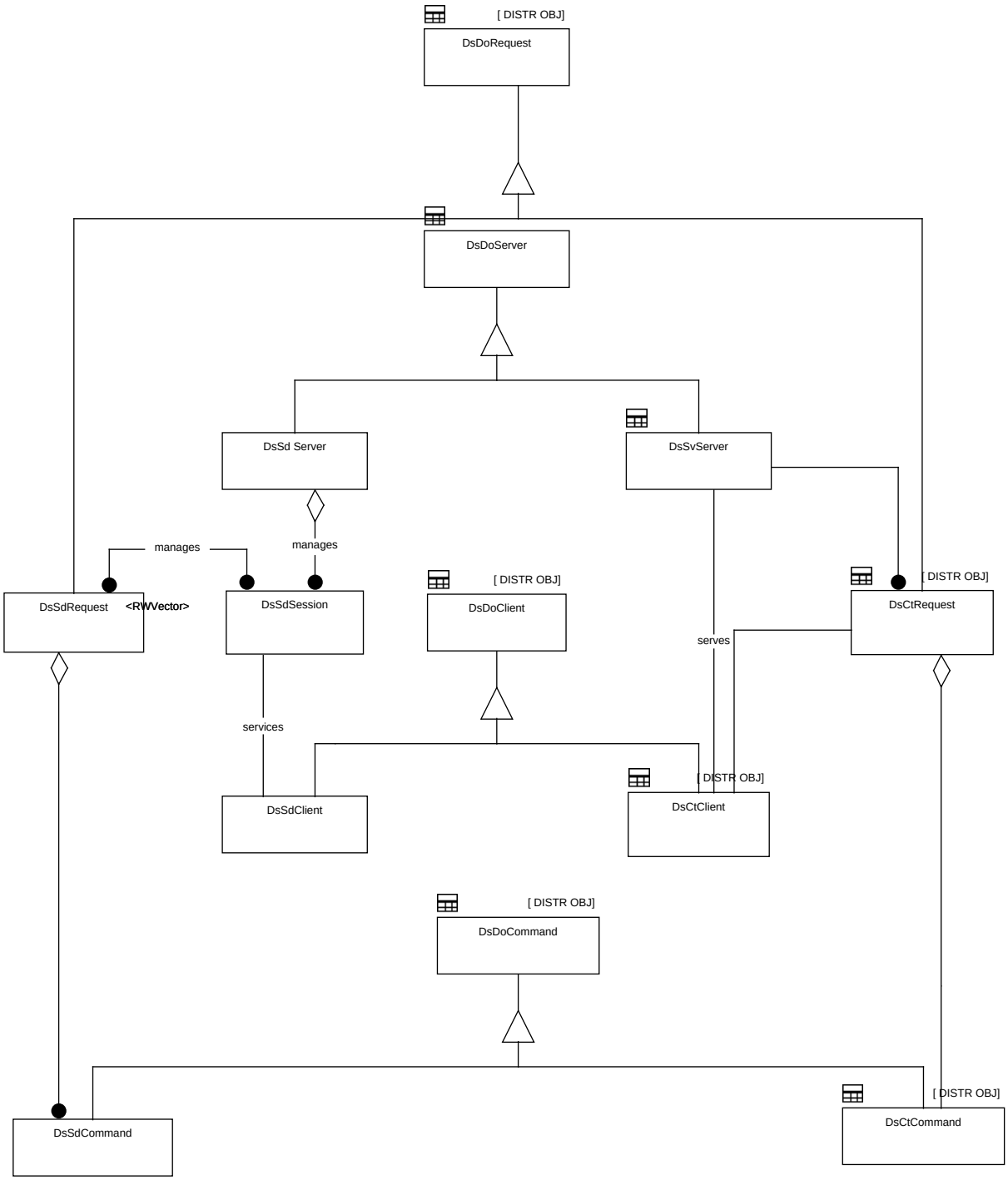


Figure 5.3-1. DsDoServer Object Model Diagram

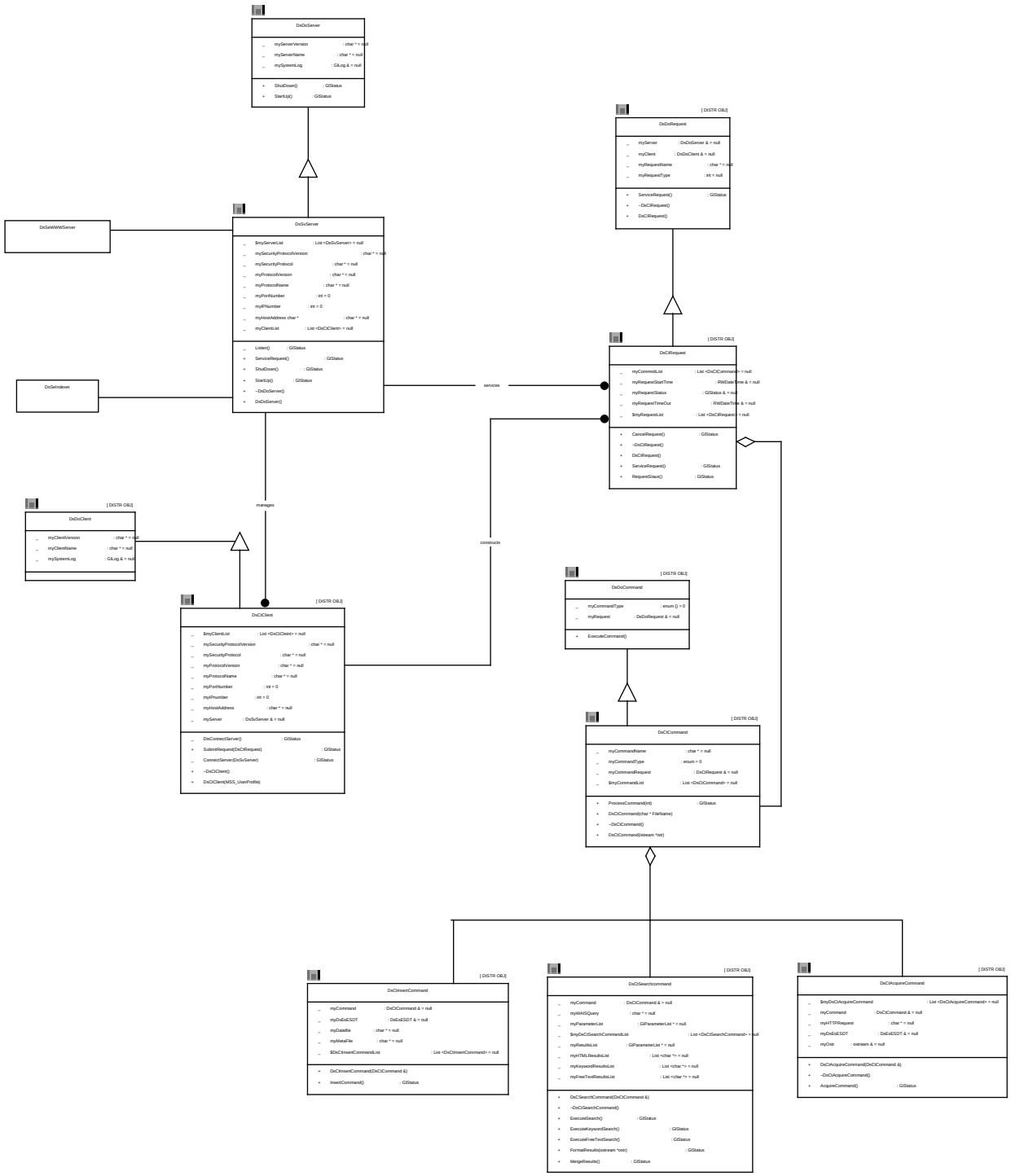


Figure 5.3-2. DsDoDocumentServer Object Model Diagram

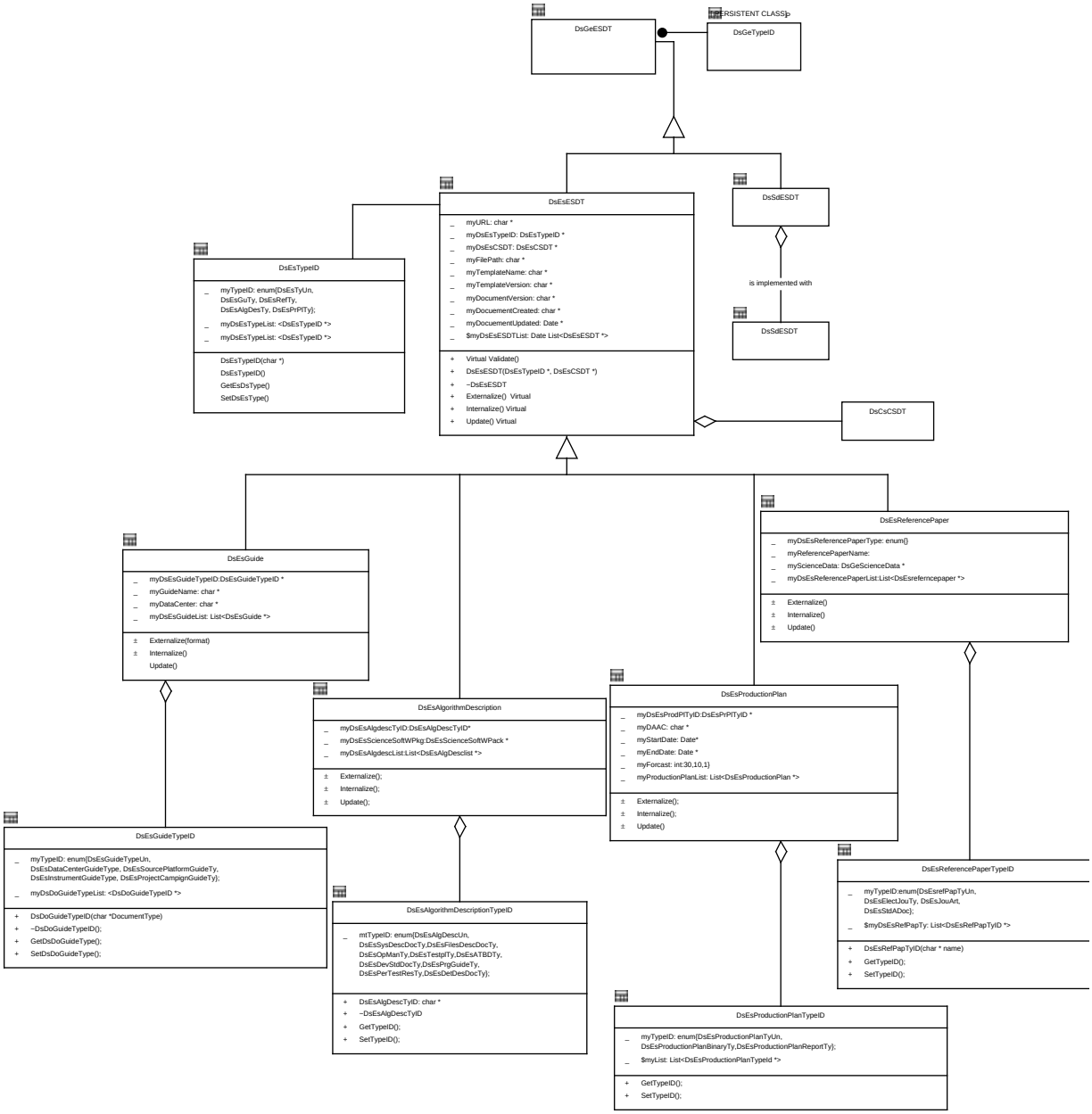


Figure 5.3-3. DsDoESDT Object Model Diagram

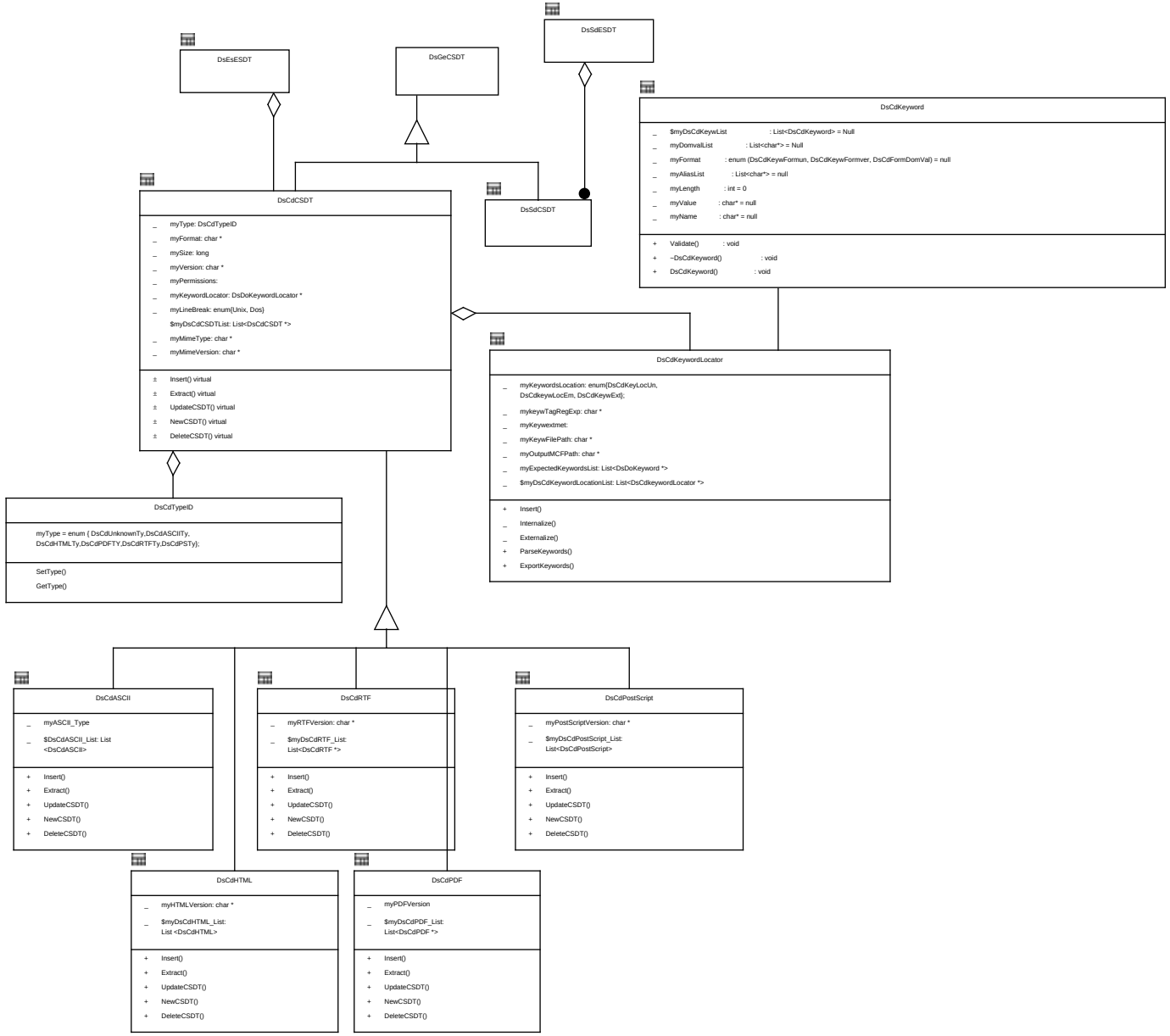


Figure 5.3-4. DsDoCSDT Object Model Diagram

5.3.1 DsCdASCII Class

Parent Class: DsCdCSDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the ASCII type for the document data.

Attributes:

$DsCdASCII_List
Data Type:
Privilege: Private
Default Value:

myASCII_Type
Data Type:
Privilege: Private
Default Value:

Operations:

DeleteCSDT
Arguments:
Return Type: Void
Privilege: Public

Extract
Arguments:
Return Type: Void
Privilege: Public

Insert
Arguments:
Return Type: Void
Privilege: Public

NewCSDT
Arguments:
Return Type: Void
Privilege: Public

UpdateCSDT
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCdASCII class has associations with the following classes:
None

5.3.2 DsCdCSDT Class

Parent Class: DsGeCSDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the CSDT of the Document ESDT.

Attributes:

$myDsCdCSDTList

myFormat
Data Type:
Privilege: Private
Default Value:

myKeywordLocator
Data Type:
Privilege: Private
Default Value:

myLineBreak
Data Type:
Privilege: Private
Default Value:

myMimeType
Data Type:

Privilege: Private
Default Value:

myMimeVersion
Data Type:
Privilege: Private
Default Value:

myPermissions
Data Type:
Privilege: Private
Default Value:

mySize
Data Type:
Privilege: Private
Default Value:

myType
Data Type:
Privilege: Private
Default Value:

myVersion
Data Type:
Privilege: Private
Default Value:

Operations:

DeleteCSDT
Arguments:
Return Type: Void
Privilege: Protected

Extract
Arguments:
Return Type: Void
Privilege: Protected

Insert
Arguments:
Return Type: Void
Privilege: Protected

NewCSDT
Arguments:
Return Type: Void
Privilege: Protected

UpdateCSDT
Arguments:
Return Type: Void
Privilege: Protected

Associations:

The DsCdCSDT class has associations with the following classes:
DsEsESDT (Aggregation)

5.3.3 DsCdHTML Class

Parent Class: DsCdCSDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the HTML CSDT type for the Document data.

Attributes:

$myDsCdHTML_List
Data Type:
Privilege: Private
Default Value:

myHTMLVersion
Data Type:
Privilege: Private
Default Value:

Operations:

DeleteCSDT
Arguments:
Return Type: Void
Privilege: Public

Extract
Arguments:
Return Type: Void
Privilege: Public

Insert
Arguments:
Return Type: Void
Privilege: Public

NewCSDT
Arguments:
Return Type: Void
Privilege: Public

UpdateCSDT
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCdHTML class has associations with the following classes:
None

5.3.4 DsCdKeyword Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:
This object represents the CSDT keyword for the document data type ESDT.

Attributes:

$myDsCdKeywList
Data Type: List<DsCdKeyword>
Privilege: Private
Default Value: Null

myAliasList
Data Type: List<char*>
Privilege: Private
Default Value: null

myDomvalList
Data Type: List<char*>
Privilege: Private
Default Value: Null

myFormat
Data Type: enum (DsCdKeywFormun, DsCdKeywFormver, DsCdFormDomVal)
Privilege: Private
Default Value: null

myLength
Data Type: int
Privilege: Private
Default Value: 0

myName
Data Type: char*
Privilege: Private
Default Value: null

myValue
Data Type: char*
Privilege: Private
Default Value: null

Operations:

DsCdKeyword
Arguments:
Return Type: void
Privilege: Public

Validate
Arguments:
Return Type: void
Privilege: Public

~DsCdKeyword
Arguments:
Return Type: void
Privilege: Public

Associations:

The DsCdKeyword class has associations with the following classes:
Class: DsCdKeywordLocator

5.3.5 DsCdKeywordLocator Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:
This object represnts the Keyword locator for the document CSDT's.

Attributes:

$myDsCdKeywordLocationList
Data Type:
Privilege: Private
Default Value:

myExpectedKeywordsList
Data Type:
Privilege: Private
Default Value:

myKeywFilePath
Data Type:
Privilege: Private
Default Value:

myKeywextmet
Data Type:
Privilege: Private
Default Value:

myKeywordsLocation
Data Type:
Privilege: Private
Default Value:

myOutputMCFPath
Data Type:
Privilege: Private
Default Value:

mykeywTagRegExp
Data Type:
Privilege: Private
Default Value:

Operations:

ExportKeywords
Arguments:
Return Type: Void
Privilege: Public

Externalize
Arguments:
Return Type: Void
Privilege: Private

Insert
Arguments:
Return Type: Void
Privilege: Public

Internalize
Arguments:
Return Type: Void
Privilege: Private

ParseKeywords
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCdKeywordLocator class has associations with the following classes:
Class: DsCdKeyword
DsCdCSDT (Aggregation)

5.3.6 DsCdPDF Class

Parent Class: DsCdCSDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the PDF CSDT type for the Document Data.

Attributes:

$myDsCdPDF_List
Data Type:
Privilege: Private
Default Value:

myPDFVersion
Data Type:
Privilege: Private
Default Value:

Operations:

DeleteCSDT
Arguments:
Return Type: Void
Privilege: Public

Extract
Arguments:
Return Type: Void
Privilege: Public

Insert
Arguments:
Return Type: Void
Privilege: Public

NewCSDT
Arguments:
Return Type: Void
Privilege: Public

UpdateCSDT
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCdPDF class has associations with the following classes:
None

5.3.7 DsCdPostScript Class

Parent Class: DsCdCSDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
Thie object represents the PostScript CSDT type for Document data.

Attributes:

$myDsCdPostScript_List
Data Type:
Privilege: Private
Default Value:

myPostScriptVersion
Data Type:
Privilege: Private
Default Value:

Operations:

DeleteCSDT
Arguments:
Return Type: Void
Privilege: Public

Extract
Arguments:
Return Type: Void
Privilege: Public

Insert
Arguments:
Return Type: Void
Privilege: Public

NewCSDT
Arguments:
Return Type: Void
Privilege: Public

UpdateCSDT
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCdPostScript class has associations with the following classes:
None

5.3.8 DsCdRTF Class

Parent Class: DsCdCSDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the RTF CSDT type for the Document data.

Attributes:

$myDsCdRTF_List
Data Type:
Privilege: Private
Default Value:

myRTFVersion
Data Type:
Privilege: Private
Default Value:

Operations:

DeleteCSDT
Arguments:
Return Type: Void
Privilege: Public

Extract
Arguments:
Return Type: Void
Privilege: Public

Insert
Arguments:
Return Type: Void
Privilege: Public

NewCSDT
Arguments:
Return Type: Void
Privilege: Public

UpdateCSDT
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCdRTF class has associations with the following classes:
None

5.3.9 DsCdTypeID Class

Parent Class: Not Applicable

Attributes:

Operations:

GetType
Arguments:

SetType
Arguments:

Associations:

The DsCdTypeID class has associations with the following classes:
DsCdCSDT (Aggregation)

5.3.10 DsCsCSDT Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:

Attributes:

None

Operations:

None

Associations:

The DsCsCSDT class has associations with the following classes:
DsEsESDT (Aggregation)

5.3.11 DsCtAcquireCommand Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
This object represents the document retrieval commands received for the document data.

Attributes:

$myDsCtAcquireCommand - List of all active acquire commands. Used mainly for fault recovery and memory management.
Data Type: List <DsCtAcquireCommand>
Privilege: Private
Default Value: null

myCommand - Reference to the associated command.
Data Type: DsCtCommand &
Privilege: Private
Default Value: null

myDsEsESDT - Reference to the document ESDT associated with the acquire command. The CSDT related to this ESDT is used to implement the extract operation.
Data Type: DsEsESDT &
Privilege: Private
Default Value: null

myHTTPRequest - HTTP Get command string. Used to identify the location of the document to return.

Data Type: char *
Privilege: Private
Default Value: null

myOstr - Output stream to write document data.
Data Type: ostream &
Privilege: Private
Default Value: null

Operations:

AcquireCommand
Arguments:
Return Type: GlStatus
Privilege: Public

DsCtAcquireCommand
Arguments: DsCtCommand &
Return Type: Void
Privilege: Public

~DsCtAcquireCommand - Object destructor - invoke ESDT destructor.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCtAcquireCommand class has associations with the following classes:
DsCtCommand (Aggregation)

5.3.12 DsCtClient Class

Parent Class: DsDoClient
Public: YesDistributed Object: Yes
Purpose and Description:
This object is the client object for the server object of Document Data server.

Attributes:

$myClientList - List of active clients.
Data Type: List <DsCtCleint>
Privilege: Private
Default Value: null

myHostAddress - Internet address of client's host.
Data Type: char *
Privilege: Private
Default Value: null

myIPnumber - IP address of the client's host.
Data Type: int
Privilege: Private
Default Value: 0

myPortNumber - TCP/IP port number of the client.
Data Type: int
Privilege: Private
Default Value: 0

myProtocolName - Name of the protocol used for the client/server connection. For external connections the protocol will be HTTP V1.0
Data Type: char *
Privilege: Private
Default Value: null

myProtocolVersion - Version of the communication protocol used by the client. For external connections the protocol will be HTTP V1.0
Data Type: char *
Privilege: Private
Default Value: null

mySecurityProtocol - Name of the security protocol used by the client.
Data Type: char *
Privilege: Private
Default Value: null

mySecurityProtocolVersion - Version of the client's security protocol.
Data Type: char *
Privilege: Private
Default Value: null

myServer - Reference to the clients associated server.
Data Type: DsSvServer &
Privilege: Private
Default Value: null

Operations:

ConnectServer - Make a connection to the server. This connection is synchronous and is established for the duration of the transaction to service the request.
Arguments: DsSvServer
Return Type: GlStatus
Privilege: Private

DisConnectServer - Disconnection of the client from the server. This operation is invoked when the external client unexpectedly quits. For example the user may select the stop button on the WWW Client interface, this breaks the TCP/IP socket and the server is notified. All outstanding service requests for the client need to be aborted.
Arguments:
Return Type: GlStatus
Privilege: Private

DsCtClient - Object constructor - for internal ECS connections the MSS_UserProfile must be specified. For external connections via HTTP, a default profile will be used which allows read access only.
Arguments: MSS_UserProfile
Return Type: Void
Privilege: Public

SubmitRequest - Submit a request to be executed by the server. The DsCtCommand and DsCtRequest structures should be constructed before this operation is invoked. The parameters described in each command should be validated prior before it is submitted to the server.
Arguments: DsCtRequest
Return Type: GlStatus
Privilege: Public

~DsCtClient - Object destructor - no specific implementation
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCtClient class has associations with the following classes:
Class: DsCtRequest constructs
Class: DsSvServer manages

5.3.13 DsCtCommand Class

Parent Class: DsDoCommand
Public: YesDistributed Object: Yes
Purpose and Description:
This object represents the client commands received for Document data.

Attributes:

$myCommandList - List of all active commands
Data Type: List <DsCtCommand>
Privilege: Private
Default Value: null

myCommandName - Name of the service to execute
Data Type: char *
Privilege: Private
Default Value: null

myCommandRequest - Associated request
Data Type: DsCtRequest &
Privilege: Private
Default Value: null

myCommandType - Command type to execute
Data Type: enum
Privilege: Private
Default Value: 0

Operations:

DsCtCommand - Constructor to read command from specified file.
Arguments: char * FileName

Return Type: Void
Privilege: Public

DsCtCommand
Arguments: istream *istr
Return Type: Void
Privilege: Public

ProcessCommand - Called to execute this command. The appropriate sub-command operation is invoked.
Arguments: int
Return Type: GlStatus
Privilege: Public

~DsCtCommand - Object destructor - no specific implementation
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCtCommand class has associations with the following classes:
DsCtRequest (Aggregation)

5.3.14 DsCtInsertCommand Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
This object represents the insert commands received for the document data.

Attributes:

$DsCtInsertCommandList
Data Type: List <DsCtInsertCommand>
Privilege: Private
Default Value: null

myCommand
Data Type: DsCtCommand &

Privilege: Private
Default Value: null

myDatafile
Data Type: char *
Privilege: Private
Default Value: null

myDsEsESDT
Data Type: DsEsESDT &
Privilege: Private
Default Value: null

myMetaFile
Data Type: char *
Privilege: Private
Default Value: null

Operations:

DsCtInsertCommand - Constructor with reference to assocaited command.
Arguments: DsCtCommand &
Return Type: Void
Privilege: Public

InsertCommand - Invoked for the insertion of a document into the document repository. The associated DsEsESDT object is called with an internalize() method to insert the metadata into the DBMS through the DBMS wrapper layer. Returns a GlStatus to indicate success or failure.
Arguments:
Return Type: GlStatus
Privilege: Public

Associations:

The DsCtInsertCommand class has associations with the following classes:
DsCtCommand (Aggregation)

5.3.15 DsCtRequest Class

Parent Class: DsDoRequest
Public: YesDistributed Object: Yes
Purpose and Description:
This object represents the requests from client to the Document Data Server.

Attributes:

$myRequestList - List of currently active requests. Used for fault recovery and memory management.
Data Type: List <DsCtRequest>
Privilege: Private
Default Value: null

myCommndList - List of the commands associated with this request.
Data Type: List <DsCtCommand>
Privilege: Private
Default Value: null

myRequestStartTime - Date and time of request submition.
Data Type: RWDateTime &
Privilege: Private
Default Value: null

myRequestStatus - Current status of request.
Data Type: GlStatus &
Privilege: Private
Default Value: null

myRequestTimeOut - Date and Time for request to time out.
Data Type: RWDateTime &
Privilege: Private
Default Value: null

Operations:

CancelRequest - Called to cancel a currently active request.
Arguments:
Return Type: GlStatus
Privilege: Public

DsCtRequest - Default constructor.
Arguments:
Return Type: Void
Privilege: Public

RequestStaus - Return the status of the currently executing request.
Arguments:
Return Type: GlStatus
Privilege: Public

ServiceRequest - Called to execute the commands associated with this request. Loop through the command list and execute the appropriate service request.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsCtRequest - Object destructor - call destructors for associated commands.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCtRequest class has associations with the following classes:
Class: DsCtClient constructs
Class: DsSvServer services

5.3.16 DsCtSearchcommand Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
This object represents the search commands received for the document data.

Attributes:

$myDsCtSearchCommandList
Data Type: List <DsCtSearchCommand>
Privilege: Private
Default Value: null

myCommand - Reference to associated command object.
Data Type: DsCtCommand &
Privilege: Private
Default Value: null

myFreeTextResultsList - The list of URLs returned as a result of a free text query submitted to the COTS search engine.
Data Type: List <char *>
Privilege: Private
Default Value: null

myHTMLResultsList - The list of URLs to be packaged in a HTML document which is returned to the WWW client across the HTTP connection.
Data Type: List <char *>
Privilege: Private
Default Value: null

myKeywordResultsList - The list of URLs returned as a result of a keyword search submitted to the DBMS wrapper layer.
Data Type: List <char *>
Privilege: Private
Default Value: null

myParameterList - The GlParameterList which represents the query to be submitted to the DBMS wrapper layer. These search parameters are extracted from the WAIS query string.
Data Type: GlParameterList *
Privilege: Private
Default Value: null

myResultsList - The results set returned from the DBMS wrapper layer. From this list the matching URLs are copied to the keyword results list.
Data Type: GlParameterList *
Privilege: Private
Default Value: null

myWAISQuery - WAIS query to be executed. The query string is copied from the QUERY_STRING environmental passed through the CGI interface call. The query may be keyword or free text.
Data Type: char *
Privilege: Private
Default Value: null

Operations:

DsCSearchCommand - Constructor with a reference to this object's associated command.
Arguments: DsCtCommand &
Return Type: Void
Privilege: Public

ExecuteFreeTextSearch - Used to search the COTS search engine for free text searches.
Arguments:
Return Type: GlStatus
Privilege: Public

ExecuteKeywordSearch - Used to submit a query to the DBMS wrapper layer. The results are stored in the results list.
Arguments:
Return Type: GlStatus
Privilege: Public

ExecuteSearch - Called to execute the search. Both a keyword and free-text search a executed as appropriate.
Arguments:
Return Type: GlStatus
Privilege: Public

FormatResults - Format the merged results into a HTML document to be returned to the WWW client across the HTTP connection. Write output to specified output stream.
Arguments: ostream *ostr
Return Type: GlStatus
Privilege: Public

MergeResults - Used to merge the results of a free text and keyword search. A list of URLS are writted to the HTML results list.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsCtSearchCommand - Object destructor - Deep distruction, all associated results lists need to be freed. Call distructor for GlParameterList and lists of URLs.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsCtSearchcommand class has associations with the following classes:
DsCtCommand (Aggregation)

5.3.17 DsDoClient Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
This object represnts the client for Document Data server.

Attributes:

myClientName - Name of the client. Used to identify the protocol and behaviour of a WWW client.
Data Type: char *
Privilege: Private
Default Value: null

myClientVersion - Version number of the client. Different WWW clients may have differnt behaviour and different versions may present a different interface.
Data Type: char *
Privilege: Private
Default Value: null

mySystemLog - Reference to the system log for exception reporting and logging.
Data Type: GlLog &
Privilege: Private
Default Value: null

Operations:

None

Associations:

The DsDoClient class has associations with the following classes:
None

5.3.18 DsDoCommand Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
This object represnts the commands for Document Data.

Attributes:

myCommandType - Type of command to execute. Relates to the service request type.
Data Type: enum {}
Privilege: Private
Default Value: 0

myRequest - Reference to the associated request.
Data Type: DsDoRequest &
Privilege: Private
Default Value: null

Operations:

ExecuteCommand - Called to invoke the appropriate service object according to the command type.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDoCommand class has associations with the following classes:
None

5.3.19 DsDoRequest Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes

Persistent Class:
Purpose and Description:
This object represents the requests for document data received vy the Docuement Data Server.

Attributes:

myClient - Reference to the associated client for this request.
Data Type: DsDoClient &
Privilege: Private
Default Value: null

myRequestName - Name of this request - taken from the GlParameterList for this request.
Data Type: char *
Privilege: Private
Default Value: null

myRequestType - Type of request to be serviced.
Data Type: int
Privilege: Private
Default Value: null

myServer - Reference to the associated server for this request.
Data Type: DsDoServer &
Privilege: Private
Default Value: null

Operations:

DsCIRequest - Object constructor - no specific implementation
Arguments:
Return Type: Void
Privilege: Public

ServiceRequest - Invoked for the execution of this service request, normally from the server after the client has constructed the request and called service request to the server.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsCIRequest - Object destructor - call the destructor of the associated commands.
Arguments:

Return Type: Void
Privilege: Public

Associations:

The DsDoRequest class has associations with the following classes:
None

5.3.20 DsDoServer Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents server receiving requests for data from Data Server.

Attributes:

myServerName - Name of the server object.
Data Type: char *
Privilege: Private
Default Value: null

myServerVersion - Version of the server running.
Data Type: char *
Privilege: Private
Default Value: null

mySystemLog - Reference to the associated log file for fault and error logging.
Data Type: GlLog &
Privilege: Private
Default Value: null

Operations:

ShutDown - Shut down the server aborting outstanding requests.
Arguments:
Return Type: GlStatus
Privilege: Public

StartUp - Start up server and return status indicaing readiness to service incomming requests.
Arguments:
Return Type: GlStatus
Privilege: Public

Associations:

The DsDoServer class has associations with the following classes:
None

5.3.21 DsEsAlgorithmDescription Class

Parent Class: DsEsESDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the Algorthimic description for the ESDT's of document type.

Attributes:

myDsEsAlgdescList
Data Type:
Privilege: Private
Default Value:

myDsEsAlgdescTyID
Data Type:
Privilege: Private
Default Value:

myDsEsScienceSoftWPkg
Data Type:
Privilege: Private
Default Value:

Operations:

Externalize
Arguments:
Return Type: Void
Privilege: Protected

Internalize
Arguments:
Return Type: Void
Privilege: Protected

Update
Arguments:
Return Type: Void
Privilege: Protected

Associations:

The DsEsAlgorithmDescription class has associations with the following classes:
None

5.3.22 DsEsAlgorithmDescriptionTypeID Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the ID of the Algorithm Description type for the ESDT's.

Attributes:

mtTypeID
Data Type:
Privilege: Private
Default Value:

Operations:

DsEsAlgDescTyID: char *
Arguments:
Return Type: Void
Privilege: Public

GetTypeID
Arguments:
Return Type: Void
Privilege: Public

SetTypeID
Arguments:
Return Type: Void
Privilege: Public

~DsEsAlgDescTyID
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsEsAlgorithmDescriptionTypeID class has associations with the following classes:
DsEsAlgorithmDescription (Aggregation)

5.3.23 DsEsESDT Class

Parent Class: DsGeESDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the ESDT's of document type.

Attributes:

$myDsEsESDTList
Data Type:

Privilege: Private
Default Value:

myDocuementCreated
Data Type:
Privilege: Private
Default Value:

myDocuementUpdated
Data Type:
Privilege: Private
Default Value:

myDocumentVersion
Data Type:
Privilege: Private
Default Value:

myDsEsCSDT
Data Type:
Privilege: Private
Default Value:

myDsEsTypeID
Data Type:
Privilege: Private
Default Value:

myFilePath
Data Type:
Privilege: Private
Default Value:

myTemplateName
Data Type:
Privilege: Private
Default Value:

myTemplateVersion
Data Type:
Privilege: Private
Default Value:

myURL
Data Type:

Privilege: Private
Default Value:

Operations:

DsEsESDT
Arguments: DsEsTypeID *, DsEsCSDT *
Return Type: Void
Privilege: Public

Externalize
Arguments:
Return Type: Void
Privilege: Public

Internalize
Arguments:
Return Type: Void
Privilege: Public

Update
Arguments:
Return Type: Void
Privilege: Public

Virtual Validate
Arguments:
Return Type: Void
Privilege: Public

~DsEsESDT
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsEsESDT class has associations with the following classes:
Class: DsEsTypeID

5.3.24 DsEsGuide Class

Parent Class: DsEsESDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the Guide information for ESDT's.

Attributes:

myDataCenter
Data Type:
Privilege: Private
Default Value:

myDsEsGuideList
Data Type:
Privilege: Private
Default Value:

myDsEsGuideTypeID
Data Type:
Privilege: Private
Default Value:

myGuideName
Data Type:
Privilege: Private
Default Value:

Operations:

Externalize
Arguments: format
Return Type: Void
Privilege: Protected

Internalize
Arguments:
Return Type: Void
Privilege: Protected

Update
Arguments:

Associations:

The DsEsGuide class has associations with the following classes:
None

5.3.25 DsEsGuideTypeID Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the ID of the Guide type.

Attributes:

myDsDoGuideTypeList
Data Type:
Privilege: Private
Default Value:

myTypeID
Data Type:
Privilege: Private
Default Value:

Operations:

DsDoGuideTypeID
Arguments: char *DocumentType
Return Type: Void
Privilege: Public

GetDsDoGuideType
Arguments:
Return Type: Void
Privilege: Public

SetDsDoGuideType
Arguments:
Return Type: Void
Privilege: Public

~DsDoGuideTypeID
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsEsGuideTypeID class has associations with the following classes:
DsEsGuide (Aggregation)

5.3.26 DsEsProductionPlan Class

Parent Class: DsEsESDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the Production Plan for the ESDT's of document type.

Attributes:

myDAAC
Data Type:
Privilege: Private
Default Value:

myDsEsProdPlTyID
Data Type:
Privilege: Private
Default Value:

myEndDate
Data Type:
Privilege: Private
Default Value:

myForcast
Data Type:
Privilege: Private
Default Value:

myProductionPlanList
Data Type:
Privilege: Private
Default Value:

myStartDate
Data Type:
Privilege: Private
Default Value:

Operations:

Externalize
Arguments:
Return Type: Void
Privilege: Protected

Internalize
Arguments:
Return Type: Void
Privilege: Protected

Update
Arguments:
Return Type: Void
Privilege: Protected

Associations:

The DsEsProductionPlan class has associations with the following classes:
None

5.3.27 DsEsProductionPlanTypeID Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the ID of the type of the Production Plan.

Attributes:

$myList
Data Type:
Privilege: Private
Default Value:

myTypeID
Data Type:
Privilege: Private
Default Value:

Operations:

GetTypeID
Arguments:
Return Type: Void
Privilege: Public

SetTypeID
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsEsProductionPlanTypeID class has associations with the following classes:
DsEsProductionPlan (Aggregation)

5.3.28 DsEsReferencePaper Class

Parent Class: DsEsESDT
Public: NoDistributed Object: No
Purpose and Description:
This object represents the Reference Papers of Document type for the ESDT's.

Attributes:

myDsEsReferencePaperList
Data Type:
Privilege: Private
Default Value:

myDsEsReferencePaperType
Data Type:
Privilege: Private
Default Value:

myReferencePaperName
Data Type:
Privilege: Private
Default Value:

myScienceData
Data Type:
Privilege: Private
Default Value:

Operations:

Externalize
Arguments:
Return Type: Void
Privilege: Protected

Internalize
Arguments:
Return Type: Void
Privilege: Protected

Update
Arguments:
Return Type: Void
Privilege: Protected

Associations:

The DsEsReferencePaper class has associations with the following classes:
None

5.3.29 DsEsReferencePaperTypeID Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the ID of the Reference Papers of ESDT's.

Attributes:

$myDsEsRefPapTy
Data Type:
Privilege: Private
Default Value:

myTypeID
Data Type:
Privilege: Private
Default Value:

Operations:

DsEsRefPapTyID
Arguments: char * name
Return Type: Void
Privilege: Public

GetTypeID
Arguments:

Return Type: Void
Privilege: Public

SetTypeID
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsEsReferencePaperTypeID class has associations with the following classes:
DsEsReferencePaper (Aggregation)

5.3.30 DsEsTypeID Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
This object represents the ID of the type of the ESDT.

Attributes:

myDsEsTypeList
Data Type:
Privilege: Private
Default Value:

myTypeID
Data Type:
Privilege: Private
Default Value:

Operations:

DsEsTypeID
Arguments: char *

DsEsTypeID
Arguments:

GetEsDsType
Arguments:

SetDsEsType
Arguments:

Associations:

The DsEsTypeID class has associations with the following classes:
Class: DsEsESDT

5.3.31 DsGeCSDT Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:

Attributes:

None

Operations:

None

Associations:

The DsGeCSDT class has associations with the following classes:
None

5.3.32 DsGeESDT Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
The Earth Science Data Type (ESDT) is a superclass for the various specializations that represent the specific data types. The ESDT organizes and provides an interface to the external and internal services. Most probably this class will be an abstract base class. If there are instances of this class it will be to support a generic type of ESDT that has no data type services other than get, put, and search.

Attributes:

None

Operations:

None

Associations:

The DsGeESDT class has associations with the following classes:
Class: DsGeTypeID

5.3.33 DsGeTypeID Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class: True
Purpose and Description:
This object uniquely identifies each ESDT's type. The type consists of a type name and a version number. Each type-version number pair is assigned a unique code. The set of all TypeIDs is stored persistently in a database.

Attributes:

None

Operations:

None

Associations:

The DsGeTypeID class has associations with the following classes:
Class: DsGeESDT

5.3.34 DsSdCSDT Class

Parent Class: DsGeCSDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
Computer Science Data Type. The CSDT is a data structure used by the data server superclass that provides common services for the different types of CSDTs. The CSDT provides the internal representation of data objects.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSdCSDT class has associations with the following classes:
DsSdESDT (Aggregation)

5.3.35 DsSdClient Class

Parent Class: DsDoClient
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:

Acts as an interface class to the CSMS subsystem to use the accounting and user profiling capabilities.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSdClient class has associations with the following classes:
Class: DsSdSession services - The DsSdSession accesses all user profile and accounting information via a DsSdClient.

5.3.36 DsSdCommand Class

Parent Class: DsDoCommand
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
An implementation of the DsCommand abstract base class that provides services necessary for data server creation and storage of command objects.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSdCommand class has associations with the following classes:
DsSdRequest (Aggregation)

5.3.37 DsSdESDT Class

Parent Class: DsGeESDT
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
The Earth Science Data Type (ESDT) is a superclass for the various specializations that represent the specific data types. The ESDT organizes and provides an interface to the external and internal services.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSdESDT class has associations with the following classes:
DsSdESDT (Aggregation)

5.3.38 DsSdESDT Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
The Earth Science Data Type (ESDT) is a superclass for the various specializations that represent the specific data types. The ESDT organizes and provides an interface to the external and internal services.

Attributes:

None

Operations:

None

Associations:

The DsSdESDT class has associations with the following classes:
None

5.3.39 DsSdRequest Class

Parent Class: DsDoRequest
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To store and track services requested by a client and used in the execution of services. The Request contains all the information that has been provided by the requester as well as any information that has been acquired during the execution of the request. It has the capability to validate the components supplied in the request and to assign a unique identifier for new requests entering the system.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSdRequest class has associations with the following classes:
Class: DsSdSession manages

5.3.40 DsSdServer Class

Parent Class: DsDoServer
Public: NoDistributed Object: No

Persistent Class:
Purpose and Description:
To provide a single point of entry for external clients and to manage and create sessions. The server is the single interface to the data server. It provides the interface for a user to establish a connection with the data server.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsSdServer class has associations with the following classes:
None

5.3.41 DsSdSession Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To manage ESDT-oriented interaction between the external client and the data server. It provides an interface for the other objects in the system to communicate with the client and manages a DsSdWorkingCollection on behalf of the client. Adds request queueing and DsSdWorkingCollection handling functionality to DsSdConnection (its super class).

Attributes:

None

Operations:

None

Associations:

The DsSdSession class has associations with the following classes:
Class: DsSdRequest manages
Class: DsSdClient services - The DsSdSession accesses all user profile and accounting information via a DsSdClient.
DsSdServer (Aggregation)

5.3.42 DsSeIndexer Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:
This object represents the COTS Technology for free text Indexing.

Attributes:

None

Operations:

None

Associations:

The DsSeIndexer class has associations with the following classes:
Class: DsSvServer

5.3.43 DsSeWWWServer Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:
This object represents the COTS Technology for HTTP connections.

Attributes:

None

Operations:

None

Associations:

The DsSeWWWServer class has associations with the following classes:
Class: DsSvServer

5.3.44 DsSvServer Class

Parent Class: DsDoServer
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
The Object represents the server object for Document Data Server.

Attributes:

$myServerList - List of currently active servers. Used for fault recovery and memory management.
Data Type: List <DsSvServer>
Privilege: Private
Default Value: null

myClientList - List of the active clients being serviced by the server.
Data Type: List <DsCtClient>
Privilege: Private
Default Value: null

myHostAddress char * - Internet address of the server host.
Data Type: char *
Privilege: Private
Default Value: null

myIPNumber - IP number of the server.
Data Type: int

Privilege: Private
Default Value: 0

myPortNumber - TCP/IP port number on the host to monitor for incoming HTTP requests.
Data Type: int
Privilege: Private
Default Value: 0

myProtocolName - Name of the communication protocol supported by the server.
Data Type: char *
Privilege: Private
Default Value: null

myProtocolVersion - Version of the communication protocol supported by the server.
Data Type: char *
Privilege: Private
Default Value: null

mySecurityProtocol - Name of the security protocol used by the server. Needs to be compatable with the client security protocol for a secure connection.
Data Type: char *
Privilege: Private
Default Value: null

mySecurityProtocolVersion - Version of the security protocol used by the server.
Data Type: char *
Privilege: Private
Default Value: null

Operations:

DsDoServer - Object constructor - no specific implementation
Arguments:
Return Type: Void
Privilege: Public

Listen - Monitors the TCP/IP port for incomming requests.
Arguments:
Return Type: GlStatus
Privilege: Private

ServiceRequest - Called by client to execute a service request.
Arguments:
Return Type: GlStatus
Privilege: Public

ShutDown - Shut down of an active server. Currently active clients will have their associated service requests aborted.
Arguments:
Return Type: GlStatus
Privilege: Public

StartUp - Start up server. Return status to calling object indicating the server is ready to service incoming requests.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsDoServer - Object destructor - destroy associated client objects
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsSvServer class has associations with the following classes:
Class: DsSeIndexer
Class: DsSeWWWServer
Class: DsCtClient manages
Class: DsCtRequest services

5.4 CSCI Dynamic Model

This section describes several scenarios for the Science Data Server CSCI. A scenario delineates the steps that occur in a particular execution through the system. The scenarios address the following topics:

5.4.1 Inserting a document

5.4.2 Searching for a document

5.4.3 Acquiring a document

5.4.4 Fault Scenario : HTTP Connection Failure

5.4.1 Inserting a Document

5.4.1.1 Summary

This scenario illustrates the classes and events which interact to insert a document and its metadata into the DDSRV. The Ingest subsystem performs metadata extraction and range checking in a manner analogous to that for science data, presenting the DDSRV with the document data file and its metadata file. The DDSRV then uses the DBMS Wrappers to index the document from its metadata file keywords. In addition, the DDSRV performs full text indexing for those document types which have a free text search defined for its ESDT. After the document has been inserted into the DDSRV a status indicating a successful insert request is returned to the Ingest Subsystem, and the document is ready for searching.

5.4.1.2 Assumptions

In this scenario, the DDSRV will be ingesting a Data Center Guide document (DsDoDataCenterGuide). The Ingest Subsystem has received a Data Center Guide document in HTML format, from its client interface. The Ingest Subsystem has extracted keywords from the document, performed range checking of the keywords using metadata configuration information, and has produced a parameter value list file (PVL) containing the document keywords. The Ingest Subsystem has established a connection with the DDSRV to complete the insertion of the HTML document, and its keyword and free text indexing.

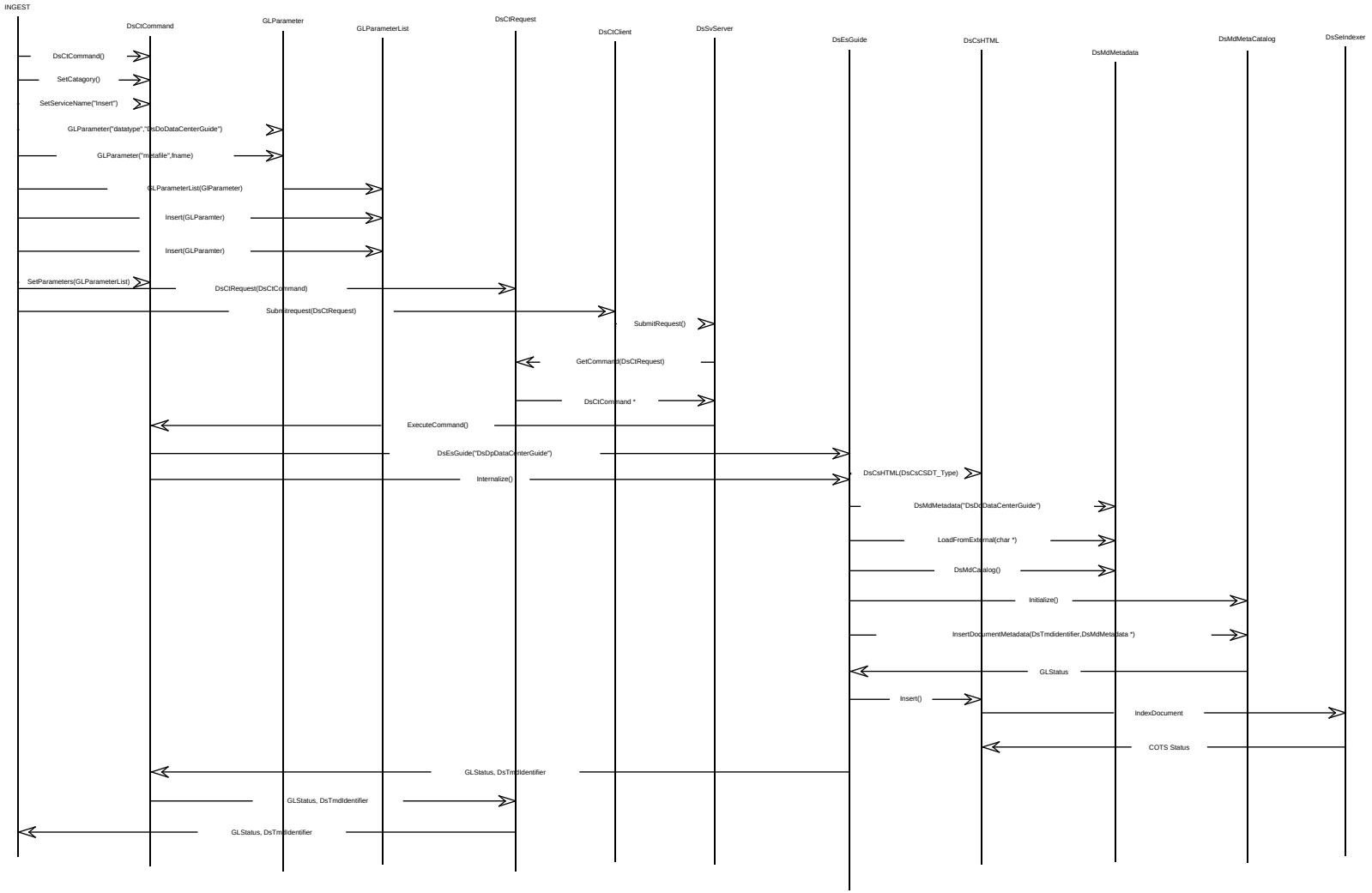


Figure 5.4-1. DDSRV_Insert Dynamic Model

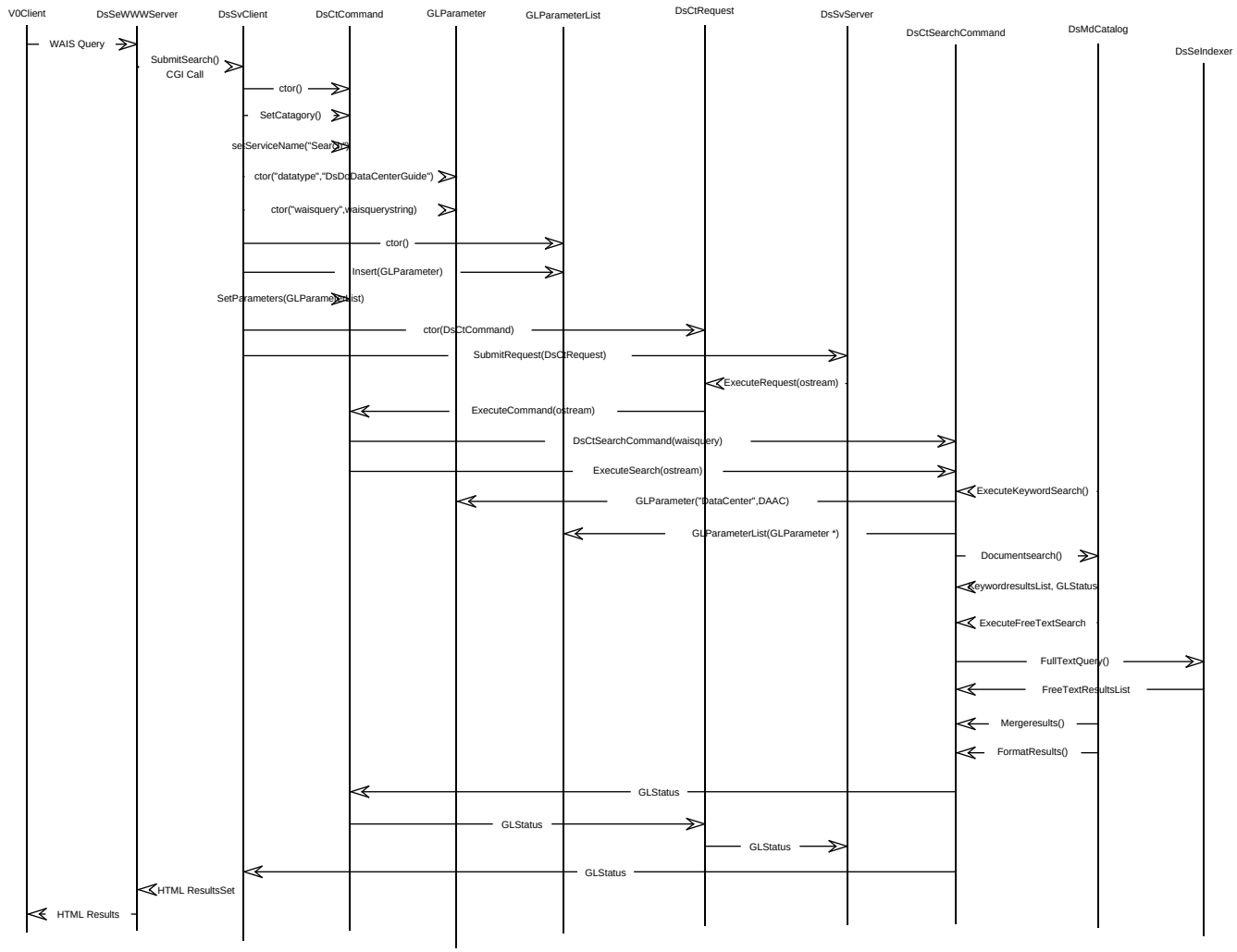


Figure 5.4-2. DDSRV_Search Dynamic Model

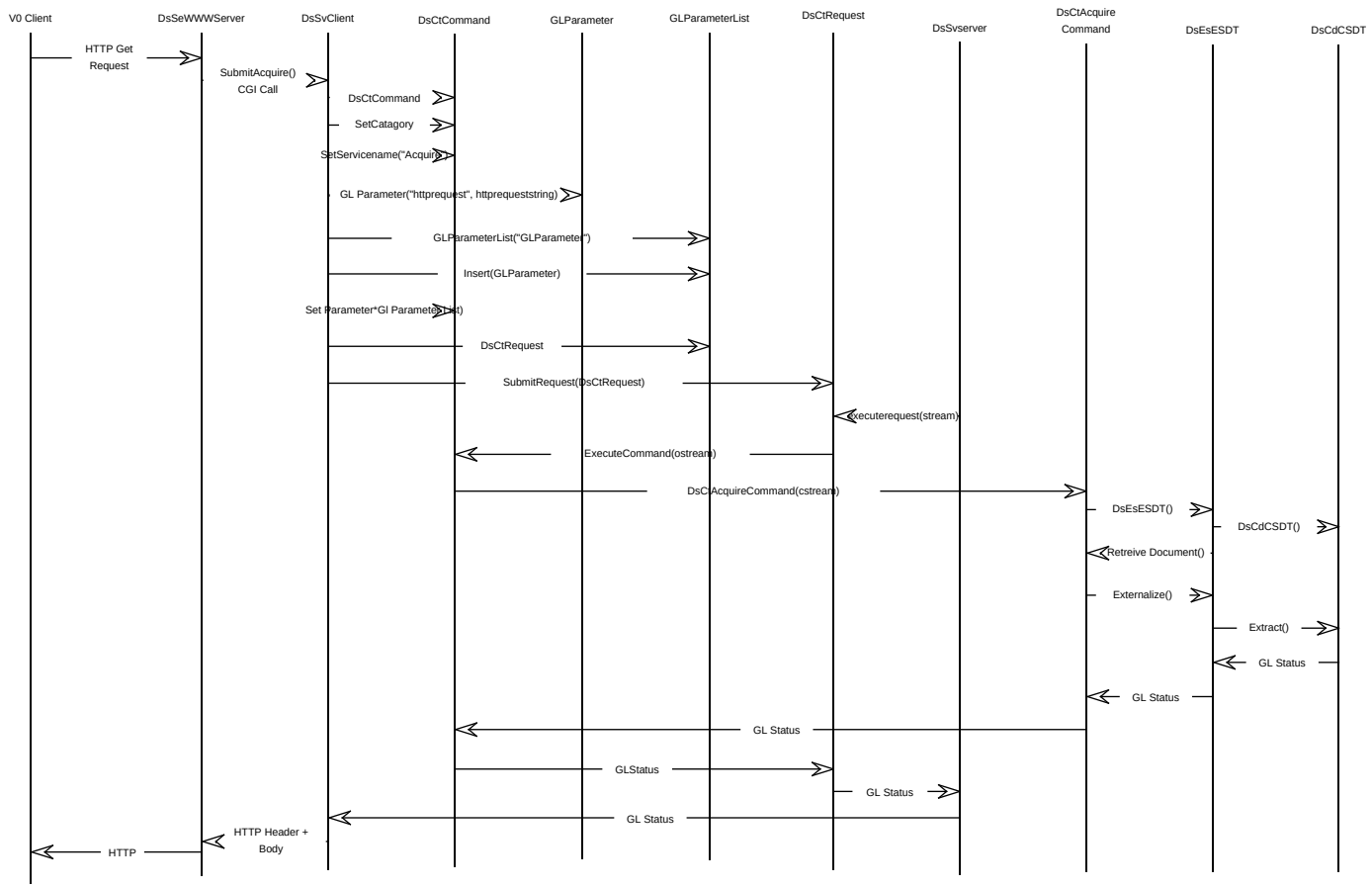


Figure 5.4-3. DDSRV_Acquire Dynamic Model

5.4.2 Searching for a Document

5.4.2.1 Summary

The DDSRV provides both a free text index and keyword index on its document holdings. The keyword index uses parameter names and values consistent with the ESDT metadata stored in the SDSRV; the free text index is performed by a COTS product. Keyword integrity between the data designs of the DDSRV and the SDSRV facilitates navigation between the various layers of the data pyramid. The scenario ends with the presentation of an HTML formatted results list of references to documents which meet the search criteria. Each document reference in the results list is presented as a hyperlink in the HTML page, and points to the relevant document that is stored in the DDSRV.

5.4.2.2 Assumptions

In this scenario, the client is requesting both a free text search and a keyword search for Data Center Guide documents. The client has formulated the query in WAIS from an interactive client capable of viewing HTML documents.

5.4.3 Acquiring a Document

5.4.3.1 Summary

The DDSRV provides storage for its documents and document metadata, and performs the on-line distribution of documents to its HTTP clients.

5.4.3.2 Assumptions

In this scenario, the client has already located the document of interest, and is requesting distribution of a Data Center Guide document via on-line HTTP connection from an interactive client capable of viewing HTML documents.

5.4.4 Fault Scenario : HTTP Connection Failure

5.4.4.1 Summary

In this scenario, during the execution of a service request by the DDSVR, the WWW Client has closed the HTTP connection. Typically this will be as a result of the user selecting the "Stop" button on the WWW Client to abort the HTTP request. Under exceptional conditions, the HTTP connection may have been closed as a result of network failure.

The HTTP server is alerted of the termination of the HTTP connection when the TCP/IP socket is broken. The DDSVR is invoked to cancel the service request. The abort command propagates through the command and request objects to the object executing the service. In this example, an acquire service is being executed by the ESDT and CSDT objects, extracting a document from the document repository.

Status information is propagated back through the call stack, to the server level. The server logs an entry in the system log to report the error and returns an HTTP error 400 to indicate the connection is broken.

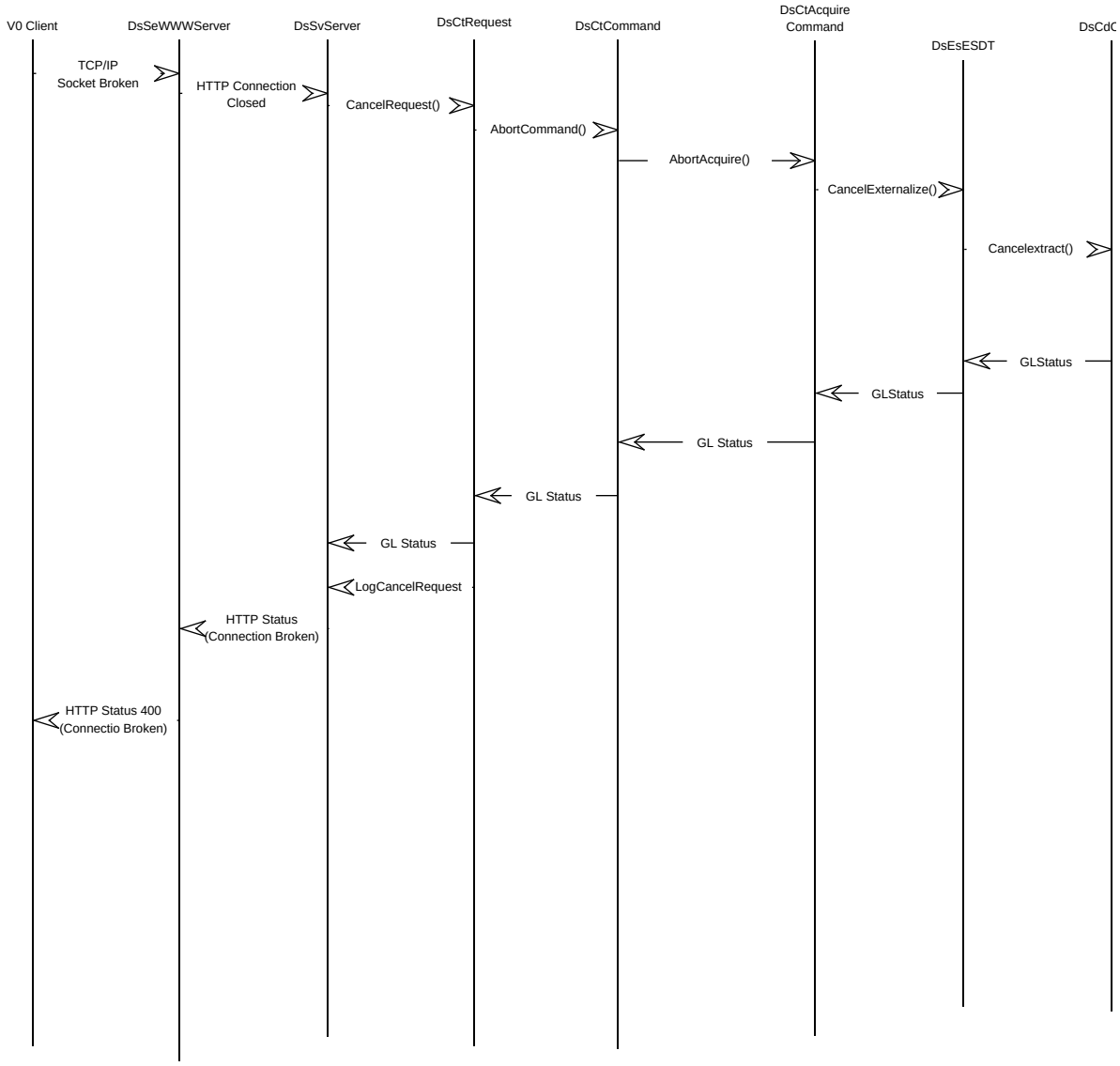


Figure 5.4-4. Fault Scenario: HTTP Connection Failure

5.4.4.2 Assumptions

A HTTP connection has been established between the WWW Client and WWW server and a HTTP transaction is under way. The DDSVR maintains sufficient information about currently active service requests to terminate those associated with the WWW Client. If possible, a HTTP Status message indicating the error will be issued by the WWW Server to the WWW Client.

5.5 CSCI Structure

5.5.1 DDSRV CSCs

The following table shows the components (CSCs) of the CSCI. Each CSC is described and designated as being custom developed code (DEV), off-the-shelf (OTS) or a combination of the two (DEV/OTS).

Table 5.5.1. DDSRV Components

CSC	Abbr.	Description	Type
DDSRV CSC	Do	The Document Data Server (DDSRV) CSC provides the base classes for the access methods and infrastructure of the DDSRV. The classes in this CSC are further specialized by the DDSRV Server CSC (Sv) and the DDSRV Client CSC (Ct) for implementing the access methods and server functions.	DEV
DDSRV Server CSC	Sv	The Document Data Server (DDSRV) Server CSC provides the primary server capabilities for this CSCI. It provides the access methods and infrastructure for handling document requests and start-up/shut down of the server.	DEV
DDSRV Client CSC	Ct	The Document Data Server (DDSRV) Client CSC provides the request and command services for clients accessing the Document Data Server. Clients connect to the DDSRV through the instantiation of these classes. The classes contained in this CSC are Public and Distributed.	DEV/OTS
DDSRV ESDT CSC	Es	The Document Data Server (DDSRV) ESDT CSC consists of the classes describing the structure and operations supported for each of the document ESDTs.	DEV/OTS
DDSRV CSDT CSC	Cd	The Document Data Server (DDSRV) Server CSDT CSC consists of the classes providing the Computer Science Data Types (CSDTs) for the document ESDTs.	DEV/OTS
DDSRV Search Engine CSC	Se	The Document Data Server (DDSRV) Search Engine CSC consists of off-the-shelf software for full-text indexing and searching of documents, and hyperlinked access to ESDT documents stored in the DDSRV.	OTS

5.5.1.1 Document Data Server (DDSRV) CSC

Purpose and Description

The Document Data Server (DDSRV) CSC provides the base classes for the access methods and infrastructure of the DDSRV. The classes in this CSC are further specialized by the DDSRV Server CSC (Sv) and the DDSRV Client CSC (Ct) for implementing the access methods and server functions.

Mapping to objects implemented by this component

DsDoServer
DsDoClient
DsDoCommand
DsDoRequest

The Document Data Server Process contains classes from this CSC.

Candidate products

N/A

ECS white paper references

N/A

5.5.1.2 Document Data Server (DDSRV) Server CSC

Purpose and Description

The Document Data Server (DDSRV) Server CSC provides the primary server capabilities for this CSCI. It provides the access methods and infrastructure for handling document requests and start-up/shut down of the server.

Mapping to objects implemented by this component

DsSvServer

This class is contained in the Document Data Server Process.

Candidate products

N/A

ECS white paper references

N/A

5.5.1.3 Document Data Server (DDSRV) Client CSC

Purpose and Description

The Document Data Server (DDSRV) Client CSC provides the request and command services for clients accessing the Document Data Server. Clients connect to the DDSRV through the instantiation of these classes. The classes contained in this CSC are Public and Distributed.

Mapping to objects implemented by this component

DsCtClient
DsCtCommand
DsCtRequest
DsCtInsertCommand
DsCtSearchCommand
DsCtAcquireCommand

The Client Applications Process is comprised of classes from this CSC.

Candidate products

N/A

ECS white paper references

N/A

5.5.1.4 Document Data Server (DDSRV) ESDT CSC

Purpose and Description

The Document Data Server (DDSRV) ESDT CSC consists of the classes describing the structure and operations supported for each of the document ESDTs.

Mapping to objects implemented by this component

DsEsESDT
DsEsTypeID
DsEsAlgorithmDescription
DsEsAlgorithmDescriptionTypeID
DsEsGuide
DsEsGuideTypeID
DsEsProductionPlan
DsEsProductionPlanTypeID
DsEsReferencepaper
DsEsReferencepaperTypeID

The Document Data Server Process contains classes from this CSC.

Candidate products

N/A

ECS white paper references

N/A

5.5.1.5 Document Data Server (DDSRV) CSDT CSC

Purpose and Description

The Document Data Server (DDSRV) Server CSDT CSC consists of the classes providing the Computer Science Data Types (CSDTs) for the document ESDTs.

Mapping to objects implemented by this component

DsCdCSDT
DsCdCSDT_TypeID
DsCdCSDT
DsCdKeywordLocator
DsCdKeyword
DsCdASCII
DsCdHTML
DsCdPDF
DsCdRTF
DsCdPostscript

The Document Repository Process is comprised of classes from this CSC.

Candidate products

N/A

ECS white paper references

N/A

5.5.1.6 Document Data Server (DDSRV) Search Engine CSC

Purpose and Description

The Document Data Server (DDSRV) Search Engine CSC consists of off-the-shelf software for full-text indexing and searching of documents, and hyperlinked access to ESDT documents stored in the DDSRV.

Mapping to objects implemented by this component

DsSeWWWServer
DsSeIndexer

The WWW Server Process contains the DsSeWWWServer class; the DsSeIndexer is contained in the Document Repository Process.

Candidate products

NetScape
CNIDR-ISite
Quadralay

ECS white paper references

N/A

5.5.2 DDSRV CI Processes

5.5.2.1 Document Data Server Process

The Document Data Server Process provides the server functions necessary for the execution of the "factory model". The Document Data Server Process consists of classes from the following CSCs :

DDSRV
DDSRV Server
DDSRV ESDT

5.5.2.2 WWW Server Process

The WWW Server process accepts client requests for searching document metadata and acquiring documents, and consists of an HTTP server and a CGI interface to the Document Data Server Process. The WWW Server process utilizes the CGI interface for mapping requests in HTTP to service requests for additional processing by the Document Data Server Process. The WWW Server Process consists of classes from the following CSC :

DDSRV Search Engine

5.5.2.3 Document Repository Process

The Document Repository Process provides persistent storage for the document Earth Science Data types. The Document Repository Process consists of classes from the following CSCs :

DDSRV CSDT
DDSRV Search Engine

5.5.2.4 Client Applications Process

The Client Applications Process provides the class library to be used by client applications for sending requests to the DDSRV. The Client Applications Process consists of classes from the following CSC :

DDSRV Client CSC

5.6 CSCI Management and Operation

The materials in the following paragraphs discuss the management and operations of software components discussed in section 4.5.

5.6.1 System Management Strategy

The DDSRV CSCI is designed to provide robust document storage, search, retrieval, and distribution services to external data providers and requestors via an encapsulated COTS product. Specifically, the design goal of the DDSRV CSCI is to always return status (successful or unsuccessful) for every received request. To accomplish that goal, the CSCI follows ECS project guidelines for:

5.6.1.1 Process startup and shutdown;

5.6.1.2 Error detection and reporting;

5.6.1.3 Fault tolerance and error recovery

5.6.1.1 Startup/Shutdown

MSS provides life-cycle services for system startup and shutdown. The DDSRV uses the services during operation. The DocumentDataServer process is started as a standalone processes. The primary role of the DocumentDataServer Process is to provide an access mechanism for search and repository requests. The WWWServer process will routinely be started at system startup. The WWWServer Process provides document index and search capabilities in addition to a communication infrastructure link for document transfer. The DocumentRepository process will also routinely be started at system startup. The DocumentRepository process responds to storage and retrieval command sent by the WWWServer process.

5.6.1.2 Error Detection and Reporting

The Document Data Server CSCI is designed for primarily automated operations with little need for operations involvement short of tuning and critical error conditions. CSS and MSS jointly provide event logging services for logging and reporting errors and faults, for browsing error/status logs, and for detecting and reporting critical errors. The Document Data Server CSCI will fully use those services during operations. Errors/status may be reported in two error logs. MSS

maintains the first log, the MSS event log. It contains errors/status of interest to operations staff to evaluate system status and to perform trend analysis. The Document Data Server CSCI maintains the second log. The Document Data Server event log contains selected errors/status from the MSS event log (for context) plus highly-detailed debug events. Software maintenance personnel use the Data Distribution event log to diagnose system and software problems in response to trouble tickets.

Non-critical errors encountered during processing that will be handled at the application level will be fully resolved and enumerated during development. Major conditions that require operator intervention and/or are considered catastrophic in the processing of requests are listed in Table 5.6-1.

Table 5.6-1. DDSRV Error Categories

Error Category	Actions to Be Taken
Initialization File/Environment Corrupt	This would be seen during a system startup process and would result in one or more executables not starting. Operations staff evaluate the condition and correct.
Unable To Allocate Disk Space	An indication disk space is running low for the COTS product. Operations staff would need to offload data.
Communications Link Dropped	This could be a serious failure of an internal component, CSCI, or an external component. Operations would need to analyze a local problem to determine in a hardware component or local CSCI has failed. Operations would coordinate external problems with the appropriate site.

5.6.1.3 Fault Tolerance and Error Recovery

Once a service request is accepted from a client (client being defined as any service requestor, not just the ECS Desktop Client), it is the Document Data Server CSCI's design goal to complete the request processing and return status (successful or unsuccessful) to the requestor. The Document Data Server CSCI is a stateless entity. All internal request and distribution queuing will be in accordance with the established http protocols of the encapsulated COTS product.

Failure scenarios with recovery methods include:

a. Failure of the Data Distribution Executable. This process is immediately restarted as a Unix standalone process. The http-based COTS product will then utilize internal capabilities to continue data distribution.

b. Failure of the processor on which the DDSRV process is running. In general, the processor automatically restarts. Restart of individual processes is handled as a combination of one or more of the above process restarts. If the processor is disabled, the disablement is detected by MSS SNMP services and a backup processor is restarted.

5.6.2 Operator Interfaces

DAAC operations personnel are provided with an X-Windows/Motif-based GUI to access operations data bases and system configuration information. Table 5.6.2-1 highlights the critical DDSRV GUI screens.

Table 5.6.2-1. Document Data Server Management GUI

GUI	Description	Data	Operations
Document Data Server Management	Primary Screen for access to the following GUIs: Document Management and COTS GUI	References to: Document Management and COTS GUIs	Logon Realizes and/or makes visible other GUIs
Document Management GUI	Allows operators to manage the documents within the Document Data Server	Document set information (contents) Documents	View contents Insert a document View a Document Edit a Document Delete a Document
COTS GUI	Provides the functionality for operators to manage the Document Data Server COTS product	TBD	TBD

5.6.3 Standard DDSRV Reports

Table 5.6.3-1. Standard Document Data Server Reports

Report Type	Report Description	Intended Audience
Inventory Update Report	Provides a record of new documents added to the DAAC, and provides summary information associated with the addition of new documents based on the reporting period which is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date. Associated information includes: number of requests received, number of successful and unsuccessful requests, number of documents archived, average documents archived per request, current number of archived documents, and average number of documents per request.	System Operator Performance Analyst Operations Supervisor DAAC Manager
Document Error	Is a summary report of the frequency of errors encountered during document processing. The reporting period is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date. Information for each unsuccessful request includes: date/ time stamp, request identifier, error code, error message.	System Operator Performance Analyst Operations Supervisor DAAC Manager

6. STMGT - Storage Management CSCI

6.1 CSCI Overview

The Storage Management CSCI is a generalization of the interfaces to the various data repositories which the Data Server Subsystem includes. The STMGT CSCI provides a single interface for making objects persistent, updating their persistent state, or removing them from persistent storage. The Data Server Subsystem will include several different types of storage services; each may offer specializations of that general interface which offer functions that are unique to that particular type of storage.

The Data Storage class interface will allow callers to specify the semantics of storage requirements which may then select a specific type or pool of storage. For example, SearchResult objects are stored temporarily for transmission to the client or for packaging on media for subsequent shipment to respective data requesters. From a resource management perspective, it is desirable to store the search results on a specific set of devices and monitor storage allocation to protect the server against shutdown due to storage overflow. One of the purposes of the data storage service class is to separate the semantics of storage requirements from the semantics of device types, storage and data management technologies, and interface syntax.

6.2 CSCI Context

Table 6.2-2 provides a mapping for the STMGT CI interfaces provided to other ECS CSCI's. The table defines the interfaces in terms of the STMGT-provided classes and the specific service to be used to accomplish that interface. The response (return parameter) from that member function is also included.

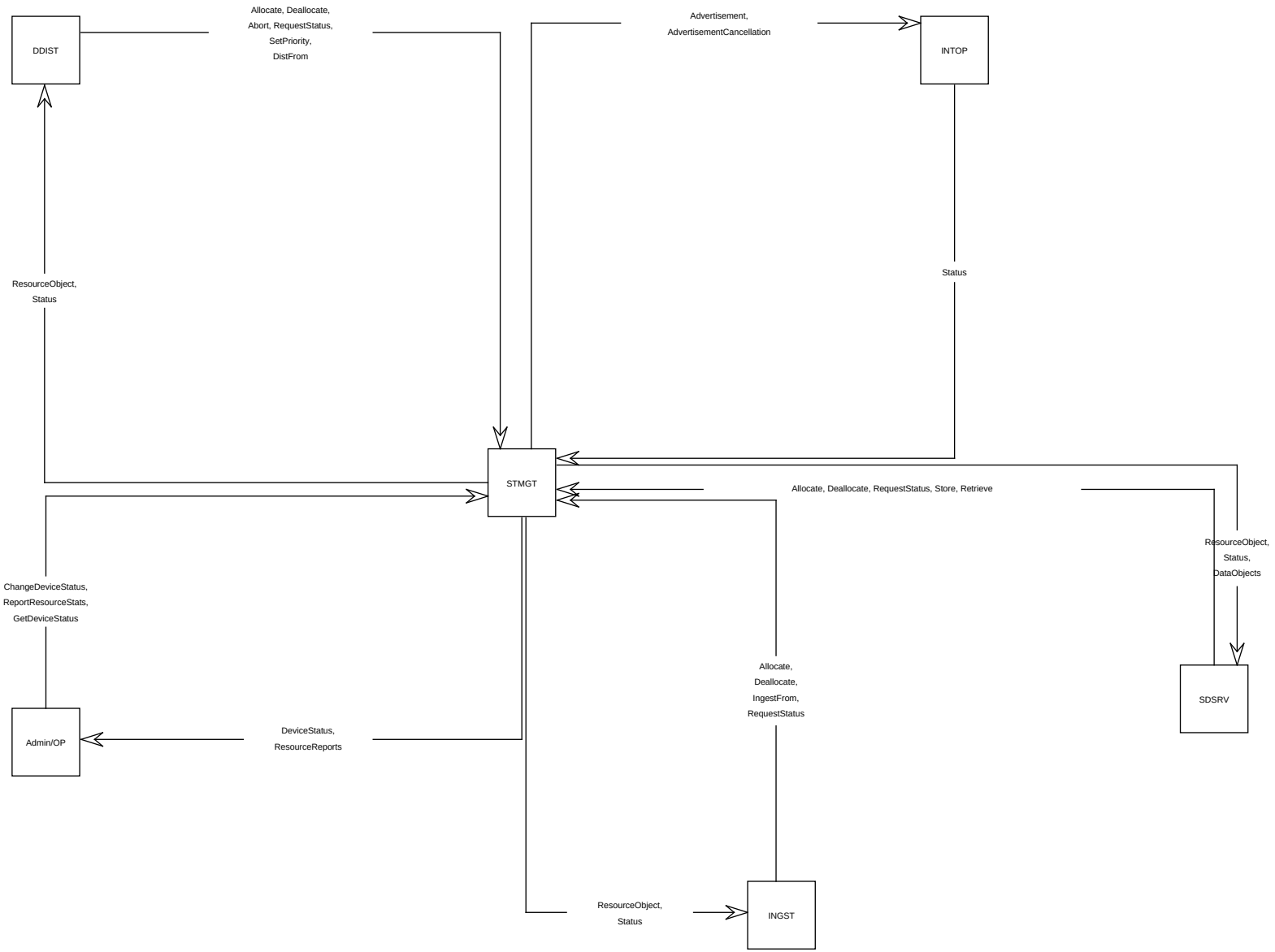


Figure 6.2-1. STMGT_events Event Flow Diagram

Table 6.2-1. STMGT_events Event Flow Summary Table

Sender	Receiver	Event Name
DDIST	STMGT	Abort
STMGT	INTOP	Advertisement
STMGT	INTOP	AdvertisementCancellation
DDIST	STMGT	Allocate
INGST	STMGT	Allocate
SDSRV	STMGT	Allocate
Admin/OP	STMGT	ChangeDeviceStatus
STMGT	SDSRV	DataObjects
DDIST	STMGT	Deallocate
INGST	STMGT	Deallocate
SDSRV	STMGT	Deallocate
STMGT	Admin/OP	DeviceStatus
DDIST	STMGT	DistFrom
Admin/OP	STMGT	GetDeviceStatus
INGST	STMGT	IngestFrom
Admin/OP	STMGT	ReportResourceStats
DDIST	STMGT	RequestStatus
INGST	STMGT	RequestStatus
SDSRV	STMGT	RequestStatus
STMGT	SDSRV	ResourceObject
STMGT	INGST	ResourceObject
STMGT	DDIST	ResourceObject
STMGT	Admin/OP	ResourceReports
SDSRV	STMGT	Retrieve
DDIST	STMGT	SetPriority
STMGT	SDSRV	Status
STMGT	INGST	Status
STMGT	DDIST	Status
INTOP	STMGT	Status
SDSRV	STMGT	Store

Table 6.2-2. CSCI Interfaces

Interface	Input Data	Output Data	Description
Advertisement Request	Service Name Service Description Service Provider	Status	This interface provides the capability to request the advertisement of services unique to the Storage Management CI
Advertisement Cancellation Request	Service Name Service Provider	Status	This interface provides the capability to request the cancellation of advertisement of services unique to the Storage Management CI
Allocate Resource Request	Requester name Resource Size	Resource Reference Status	This interface provides the capability to allocate resources.
Backup Archive Request	Archive media	Status	This interface provides the capability to operations personnel to request the backup of archive data.
Deallocate Resource Request	Allocation Identifier	Status	This interface provides the capability to deallocate previously allocated resources.
Device Control Request	Storage Device Identifier Action	Status	This interface provides operations personnel the capability to control the devices managed and monitored by storage management services.
Dismount Request	Archive volume	Status	This interface provides the capability to operations personnel to request the dismount of specific media.
Event Registration Request	Event Description Requester Identifier	Event Notification	This interface provides the capability to request notification of specific archive related events.
Insert Request	Data References Priority	Status	This interface provides the capability to request the insertion of data objects from archival storage.
Log Data Access Request	Log Identifier	Log Data Status	This interface provides the capability to request access to log data collected by the Storage Management CI
Mount Request	Archive volume	Status	This interface provides the capability to operations personnel to request the mount of specific media.
Reserve Resource Request	Resource Identifier Requester Identifier Priority	Status	This interface provides the capability to reserve specific resources managed by the Storage Management CI.
Restore Archive Request	Archive media	Status	This interface provides the capability to operations personnel to request the restoration of archived data which has been backed up.
Retrieve Request	Data Reference Pri-ority Profile Info Size	Data Objects Status	This interface provides the capability to request the retrieval of data objects from archival storage.
Status Request	Request Identifier	Status Information	This interface provides the capability to request the status of previously submitted retrieve or insert requests.

6.3 CSCI Object Model

This section provides an object model for the Storage Management CSCI. The model is depicted in Figure 6.3-1. Each of the objects in this figure is described briefly in the subsequent paragraphs. The attributes shown in the figure are described in the ECS Data Dictionary (305-CD-018 Volume 18). Other objects that may appear in the object model, but not described in the following section are allocated to other CSCIs and are replicated here for the purpose of completeness.

General Notes: In the following object class descriptions, classes always inherit attributes, operations, and associations from their parent classes. Where all attributes in the current design are inherited from the parent class, this is indicated by the text "*All Attributes inherited from the parent class*" (and analogously for operations). If a derived class has additional attributes of its own, those new attributes are listed, but the attributes from the parent class are not repeated (analogously for operations).

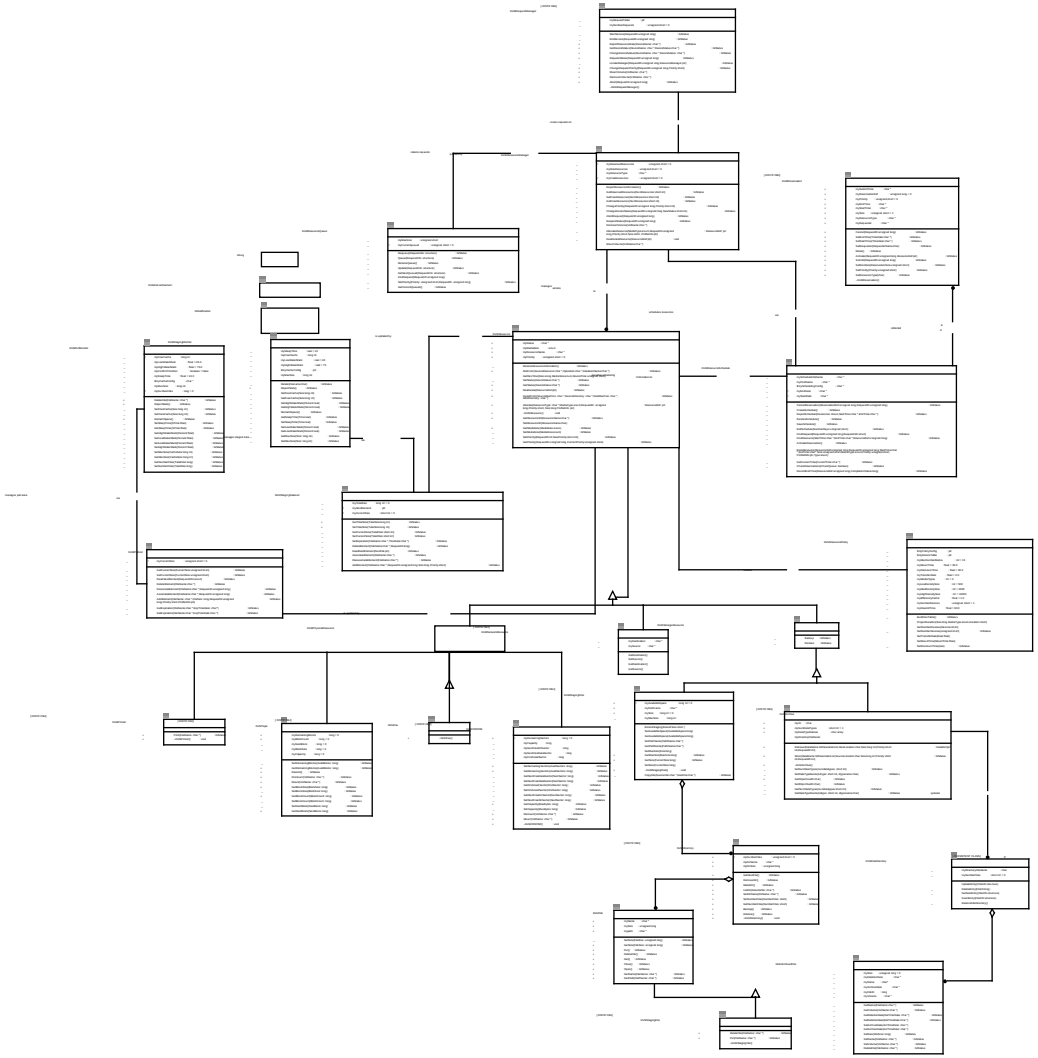


Figure 6.3-1. dsst Object Model Diagram

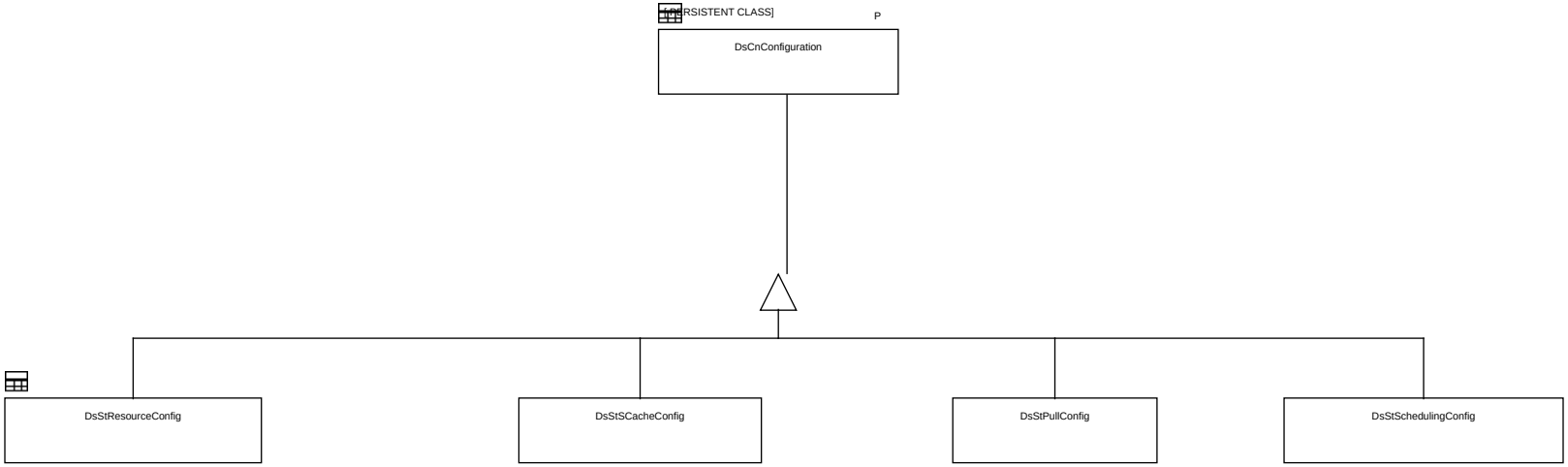


Figure 6.3-2. dsstconfig Object Model Diagram

6.3.1 DsCnConfiguration Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class: True
Purpose and Description:
Manages configuration parameters for various data server components in a token-value style file. Often specialized where needed to provide higher level access to configuration parameters.

Attributes:

None

Operations:

None

Associations:

The DsCnConfiguration class has associations with the following classes:
None

6.3.2 DsSdAdvertisement Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To provide service description and service provider information to the interoperability layer. The Advertisement is a description of a service that the data server provides.

Attributes:

None

Operations:

None

Associations:

The DsSdAdvertisement class has associations with the following classes:
None

6.3.3 DsStArchive Class

Parent Class: DsStStorageResource
Public: YesDistributed Object: Yes
Persistent Class:
Purpose and Description:
To provide storage for persistent data. The Archive is the repository for all the permanently stored data in the DataServer.

Attributes:

myDataTypeNames - This attribute identifies the data types serviced by this archive.
Data Type: char array
Privilege: Public
Default Value:

myID - This is the ID of the Archive
Data Type: char
Privilege: Public
Default Value:

myNumDataTypes - This attribute indicates the number of data types serviced by this archive.
Data Type: short int
Privilege: Public
Default Value: 1

Operations:

GetDataTypeName - This operation gets the names of the data types serviced by this archive.
Arguments: ndtype: short int, dtypename:char
Return Type: GlStatus
Privilege: Private

GetID - This operation gets the name of this archive.
Arguments: archiveID:char
Return Type: GlStatus
Privilege: Public

GetNumDataTypes - This operation gets the number of data types serviced by the archive.
Arguments: numdatatypes:short int
Return Type: GlStatus
Privilege: Private

Retrieve - This service retrieves specified files from the archive.
Arguments: DataName:GlParameterList,DestLocation:char,Size:long int,Priority:short int,RequestID:int
Return Type: DataRef:ptr
Privilege: Public

```
PDL://  Log retrieval request received
//  While (there are more files in the request and no errors)
//     Get the next file in the request
//     Initialize the data reference
//     If the requested file is in staging cache
//        If destination location is a private workarea
//           Copy file from cache to destination location
//        Set the data reference for file
//     else
//        Retrieve file from deep archive
//        If retrieval was not successful
//          Alert/Notify Operations of failure
//          Log file retrieval failure
//          Assign appropriate error number
//          If failure requires marking hardware equipment off-line
//            Mark appropriate hardware equipment off-line
//        else
//          Perform checksum on file
//          Compare checksum obtained to metadata checksum value
//          If checksums don't compare favorably
//              Alert/Notify Operations personnel
//              Log checksum error for file
//         Assign appropriate error number
//          else
//              If destination location is a private workarea
```

```
//          Copy file to destination location
//            else
//              Get the current amount of available disk space in cache
//              while (there is insufficient disk space in staging cache)
//                wait for a small period of time
//
//              Copy file to staging disk cache
//              Add an entry in the staging disk cache list for the file
//              Set the current size of available staging cache
//            Set the data reference
//
//  Log retrieval request completed
//  return (operation status)
```

SetDataTypeName - This operation sets the name(s) of the data types serviced by this archive.
Arguments: ndtype: short int, dtypename:char
Return Type: GlStatus
Privilege: Private

SetID - This operation sets the name of the archive.
Arguments: archiveID:char
Return Type: GlStatus
Privilege: Private

SetNumDataTypes - This operation sets the number of data types serviced by this archive.
Arguments: numdatatypes: short int
Return Type: GlStatus
Privilege: Private

Store - This operation stores data files into the archive.
Arguments: DataName:GlParameterList,SourceLocation:char,Size:long int,Priority:short int,RequestID:int
Return Type: GlStatus
Privilege: Public
PDL:

```
//  Log store request received
//  While (there are more files in the request)
//     Get the next file in the request
//     Archive the file
//     If archive was successful
//        Perform checksum on file
//        Log successful archive of file
//        Place entry for file in file directory
//     else
//        Log file archive failure
```

```
//        Alert/Notify Operations
//         If failure reason requires marking hardware equipment off-line
//            Mark appropriate hardware as off-line
//         Assign appropriate error number
//
//  Log store request completed
//  return  (operation status)
```

~DsStArchive - This is the destructor for the archive object.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsStArchive class has associations with the following classes:
Class: DsStFileDirectory updates

6.3.4 DsStArchivedFile Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description: This class provides information on files which have been archived. Information contained on archived files includes file name, archive data, deletion date, file size, and volume name.

Attributes:

myArchiveDate - This attribute indicates the time and date that the file was archived
Data Type: char *
Privilege: Private
Default Value:

myDeletionDate - This attribute indicates the deletion time and date for the archived file. For most files, there is no deletion time.
Data Type: char *
Privilege: Private
Default Value:

myFileID - This attribute is a unique file identifying number.
Data Type: long

Privilege: Private
Default Value:

myName - This attribute indicates the name of the archived file.
Data Type: char*
Privilege: Private
Default Value:

mySize - This attribute indicates the size (in KBYTES) of the archived file.
Data Type: unsigned long
Privilege: Private
Default Value: 0

myVolume - This attribute indicates the name of the tape volume on which the file was archived.
Data Type: char *
Privilege: Private
Default Value:

Operations:

DeleteFile - This operation deletes the file from the appropriate volume in the archive.
Arguments: FileName: char *
Return Type: GlStatus
Privilege: Private

GetArchiveDate - This operation gets the time and date that the file was archived.
Arguments: ArcTimeDate: char *
Return Type: Void
Privilege: Private

GetDeletionDate - This operation gets the deletion time and date for the archived file.
Arguments: DelTimeDate: char *
Return Type: GlStatus
Privilege: Private

GetName - This operation gets the name of the archived file.
Arguments: FileName:char *
Return Type: GlStatus
Privilege: Private

GetVolume - This operation gets the name of the tape volume on which the file was archived.

Arguments: VolName:char *
Return Type: GlStatus
Privilege: Private

SetArchiveDate - This operation sets the time and date that the file was archived.
Arguments: ArcTimeDate: char *
Return Type: Void
Privilege: Private

SetDeletionDate - This operation sets the deletion time and date for the archived file.
Arguments: DelTimeDate:char *
Return Type: GlStatus
Privilege: Private

SetName - This operation sets the name of the archived file.
Arguments: FileName: char *
Return Type: GlStatus
Privilege: Private

SetSize - This operation sets the size of the file.
Arguments: FileSize: long
Return Type: GlStatus
Privilege: Private

SetVolume - This operation sets the name of the tape volume on which the file was archived.
Arguments: VolName: char *
Return Type: GlStatus
Privilege: Private

Associations:

The DsStArchivedFile class has associations with the following classes:
DsStFileDirectory (Aggregation)

6.3.5 DsStCDROM Class

Parent Class: DsStPhysicalResource
Public: YesDistributed Object: Yes
Purpose and Description:
This class provides an interface to the CD-ROM resource. The Ingest Client to Storage

Management can use CDROM devices for ingesting data. The Data Distribution Client to Storage Managment can use these same resources for distributing data to requesting users.

Attributes:

myCapacity - This attribute indicates the capacity (in KBYTES) of the media currently mounted.
Data Type: long
Privilege: Private
Default Value:

myFirstUsedSector - This attribute indicates the first used sector on the volume currently mounted in the CDROM.
Data Type: long
Privilege: Private
Default Value:

myNextFreeDataSector - This attribute indicates the next free data sector on the volume currently mounted in the CDROM.
Data Type: long
Privilege: Private
Default Value:

myNextFreeDirSector - This attribute indicates the next free directory sector on the volume currently mounted in the CDROM.
Data Type: long
Privilege: Private
Default Value:

myRemainingSectors - This attribute indicates the number of unused (i.e., unwritten) sectors remaining on the media.
Data Type: long
Privilege: Public
Default Value: 0
This is a Derived Attribute.

Operations:

Dismount - This public operation dismounts the specified volume from a CDROM resource.
Arguments: VolName: char *
Return Type: GlStatus
Privilege: Public

GetCapacity - This operation gets the capacity of the volume currently mounted in the CDROM resource.
Arguments: MaxBytes: long
Return Type: GlStatus
Privilege: Public

GetFirstUsedSector - This operation gets the value of the first used sector on the volume currently mounted in the CDROM.
Arguments: FirstSector: long
Return Type: GlStatus
Privilege: Private

GetNextFreeDataSector - This operation gets the next free data sector on the volume currently mounted in the CDROM.
Arguments: NextSector: long
Return Type: GlStatus
Privilege: Private

GetNextFreeDirSector - This operation gets the next free directory sector on the volume currently mounted in the CDROM resource.
Arguments: NextSector: long
Return Type: GlStatus
Privilege: Private

GetRemainingSectors - This public service gets the number of free unwritten sectors on the media currently mounted in the allocated CDROM device.
Arguments: AvailSectors: long
Return Type: GlStatus
Privilege: Public

Mount - This public operation mounts the specified volume on the CDROM resource.
Arguments: VolName: char *
Return Type: GlStatus
Privilege: Public

SetCapacity - This operation sets the capacity (in KBYTES) of the media currently mounted in the CDROM resource.
Arguments: MaxBytes: long
Return Type: GlStatus
Privilege: Private

SetFirstUsedSector - This operation sets the first used sector used on the volume currently mounted in the CDROM.
Arguments: FirstSector: long

Return Type: GlStatus
Privilege: Private

SetNextFreeDataSector - This operation sets the value of the next free data sector of the volume currently mounted in the CDROM.
Arguments: NextSector: long
Return Type: GlStatus
Privilege: Private

SetNextFreeDirSector - This operation sets the value of the next free sector on the volume currently mounted in the CDROM resource.
Arguments: NextSector: long
Return Type: GlStatus
Privilege: Private

SetRemainingSectors - This operation sets the number of free and unwritten sectors remaining on the media currently mounted on the CDROM device.
Arguments: AvailSectors: long
Return Type: GlStatus
Privilege: Public

~DsStCDROM - This is the destructor for the CDROM object.
Arguments:
Return Type: void
Privilege: Public

Associations:

The DsStCDROM class has associations with the following classes:
None

6.3.6 DsStDirectory Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
This class provides the capability to partition an allocated workarea on staging disk. The allocated workarea can be made to have the appearance of an entire disk.

Attributes:

myDirName - This attribute indicates the name of the directory in the staging disk workarea.
Data Type: char *
Privilege: Public
Default Value:

myDirSize - This attribute indicates the current size (in KBYTES) of the directory created in the staging disk workarea.
Data Type: unsigned long
Privilege: Public
Default Value:

myNumberFiles - This attribute indicates the number of files that currently exist in the directory.
Data Type: unsigned short
Privilege: Public
Default Value: 0
This is a Derived Attribute.

Operations:

Backup - This operation performs a backup of the directory.
Arguments:
Return Type: GlStatus
Privilege: Public

GetNextFile - This operation gets the name of the next file in the directory created in the workarea on staging disk.
Arguments:
Return Type: GlStatus
Privilege: Public

GetNumberFiles - This operation gets the number of files in the directory.
Arguments: NumberFiles: short
Return Type: GlStatus
Privilege: Public

ListDir - This operation provides information about files in the directory.
Arguments: ResultsFile: char *
Return Type: GlStatus
Privilege: Public

MakeDir - This operation creates a directory in the workarea.
Arguments:
Return Type: GlStatus
Privilege: Public

RemoveDir - This operation removes a previously created directory from the workarea on staging disk.
Arguments:
Return Type: GlStatus
Privilege: Public

Restore - This operation restores a backed up directory to the state it was in when the backup of the directory was performed.
Arguments:
Return Type: GlStatus
Privilege: Public

SetDirName - This operation sets the name of a directory which is created in an allocated workarea on staging disk.
Arguments: DirName: char *
Return Type: GlStatus
Privilege: Public

SetNumberFiles - This operation sets the current number of files in the directory.
Arguments: NumberFiles: short
Return Type: GlStatus
Privilege: Public

~DsStDirectory - This is the destructor for the DsStDirectory object.
Arguments:
Return Type: void
Privilege: Public

Associations:

The DsStDirectory class has associations with the following classes:
DsStStagingDisk (Aggregation)

6.3.7 DsStFax Class

Parent Class: DsStPhysicalResource
Public: YesDistributed Object: Yes
Purpose and Description:
This class provides an interface to FAX resources managed by the Storage Management CI. This class is scheduled for full development in Release B.

Attributes:

All Attributes inherited from parent class

Operations:

~DsStFax - This is the destructor for the Fax object.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsStFax class has associations with the following classes:
None

6.3.8 DsStFile Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description: This class provides the capability to manipulate allocated working storage at a file level. Common file services provided by this class include open, close, read, write, and delete.

Attributes:

myName - This attribute indicates the name of the file.
Data Type: char *
Privilege: Public
Default Value:

mySize - This attribute indicates the size of the file (in KBYTES).
Data Type: unsigned long
Privilege: Public
Default Value:

mypath - This attribute indicates the full path name of the file.
Data Type: char *
Privilege: Public
Default Value:

Operations:

Close - This operation closes files which have been previously opened.
Arguments:
Return Type: GlStatus
Privilege: Public

DeleteFile - This operation deletes the file from staging disk.
Arguments:
Return Type: GlStatus
Privilege: Public

Get - This operation reads a single record of the file.
Arguments:
Return Type: GlStatus
Privilege: Public

GetName - This operation gets the name of the staging disk file.
Arguments: FileName: char *
Return Type: GlStatus
Privilege: Public

GetPath - This operation gets the path name for the staging disk file.
Arguments: PathName: char *
Return Type: GlStatus
Privilege: Public

GetSize - This operation gets the size of the file.
Arguments: FileSize: unsigned long
Return Type: GlStatus
Privilege: Public

Open - This operation opens files which have been created in an allocated workarea on staging disk.
Arguments:
Return Type: GlStatus
Privilege: Public

Put - This operation writes a single record to the file.
Arguments:
Return Type: GlStatus
Privilege: Public

SetSize - This operation sets the size of a file which has been created in an allocated workarea on staging disk.
Arguments: FileSize: unsigned long
Return Type: GlStatus
Privilege: Private

Associations:

The DsStFile class has associations with the following classes:
DsStDirectory (Aggregation)

6.3.9 DsStFileDirectory Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class: True
Purpose and Description:
To provide a collection of information on files which have been archived. Information contained in the file directory includes location of data object, size of data object and archive date of data object. The File Directory is used by the Storage Management CI to help insure that all archived data objects are present and accounted for.

Attributes:

myDirectoryFileName - This is the name of the external file that holds file directory information.
Data Type: char
Privilege: Private
Default Value:

myNumberFiles - This attribute indicates the number of current files which have been archived and are resident in the archive.
Data Type: short int
Privilege: Private
Default Value: 0

Operations:

DeleteEntry - This service will delete an entry from the file directory.
Arguments: FileID:long
Return Type: Void
Privilege: Private

GetNextEntry - This service will get the next entry in the file directory.
Arguments: FileInfo:structure
Return Type: Void
Privilege: Private

InsertEntry - This service will insert an entry in the file directory. The file directory retains information about all files which have been archived.
Arguments: FileInfo:structure
Return Type: GlStatus
Privilege: Private

RestoreFileDirectory - This service will restore a file directory that has been lost, deleted, corrupted by going back to a previous checkpointed copy.
Arguments:
Return Type: Void
Privilege: Private

UpdateEntry - This service will update an entry of the file directory.
Arguments: FileInfo:structure
Return Type: GlStatus
Privilege: Private

Associations:

The DsStFileDirectory class has associations with the following classes:
Class: DsStArchive updates

6.3.10 DsStNetworkResource Class

Parent Class: DsStResource
Public: YesDistributed Object: Yes
Purpose and Description:
This class provides a push/pull interface to the network for transferring data for both ingest and data distribution operations.

Attributes:

myDestination - This attribute identifies the destination machine for the network data transfer.
Data Type: char *
Privilege: Private
Default Value:

mySource - This attribute identifies the source machine for the network data transfer.
Data Type: char *
Privilege: Private
Default Value:

Operations:

GetDestination - This operation gets the name of the destination machine for the network data transfer.
Arguments: nodename:char*
Return Type: GlStatus
Privilege: Public

GetSource - This operation gets the name of the source machine for the network data transfer.
Arguments: nodename:char*
Return Type: GlStatus
Privilege: Public

SetDestination
Arguments: nodename:char*
Return Type: GlStatus
Privilege: Public

SetSource - This operation sets the name of the source machine for the network data transfer.
Arguments: nodename:char*

Return Type: GlStatus
Privilege: Public

Associations:

The DsStNetworkResource class has associations with the following classes:
None

6.3.11 DsStPhysicalResource Class

Parent Class: DsStResource
Public: NoDistributed Object: No
Purpose and Description:
This class provides a generic interface to all services provided by the physical ingest and data distribution resources of the Data Server.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsStPhysicalResource class has associations with the following classes:
None

6.3.12 DsStPrinter Class

Parent Class: DsStPhysicalResource
Public: YesDistributed Object: Yes
Purpose and Description:
This class provides an interface to the resource pool of printers used by data distribution to produce shipping labels, packing slips and media labels.

Attributes:

All Attributes inherited from parent class

Operations:

Print - This operation writes text to the printer.
Arguments: FileName: char *
Return Type: GlStatus
Privilege: Public

~DsStPrinter - This operation destroys the Printer object.
Arguments:
Return Type: void
Privilege: Public

Associations:

The DsStPrinter class has associations with the following classes:
None

6.3.13 DsStPullConfig Class

Parent Class: DsCnConfiguration
Public: NoDistributed Object: No
Purpose and Description:
This class is a specialization of the DsCnConfiguration base class. It manages configuration parameters for the designated electronic "pull" area on magnetic disk.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsStPullConfig class has associations with the following classes:
None

6.3.14 DsStPullList Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This class provides a mechanism for tracking the contents of the area on magnetic disk designated as the electronic "pull" area. The dynamic record of the data files currently in the pull area allows the Data Server to more efficiently and more quickly provide access to files which users have indicated they desire to electronically pull to their own workstations/ workareas.

Attributes:

myCurrentSize - This attribute indicates the size (in KBYTES) of all files currently in the distribution "pull" list (i.e., those data files that will be pulled by their respective requester).
Data Type: unsigned short
Privilege: Private
Default Value: 0
This is a Derived Attribute.

Operations:

AddElement - This operation adds a file to the distribution "pull" list.
Arguments: FileName: char *,FileSize: long,RequestID:unsigned long,Priority:short,ProfileInfo:ptr
Return Type: GlStatus
Privilege: Private

AssociateElement - This operation searches the distribution "pull" list for a specific file that has been previously distributed. If the file is found, the reference count for the file is incremented by 1.
Arguments: FileName:char *,RequestID:unsigned long
Return Type: GlStatus
Privilege: Private

DeleteElement - This operation removes the entry associated with a specific file from the distribution "pull" list.
Arguments: FileName:char *
Return Type: GlStatus
Privilege: Private

DissociateElement - This operation decrements the reference count for a specific file and a specific user in the distribution "pull" list.
Arguments: FileName:char *,RequestID:unsigned long
Return Type: GlStatus
Privilege: Private

GetCurrentSize - This operation gets the current total size (in KBYTES) of all files in the distribution "pull" list.
Arguments: CurrentSize:unsigned short
Return Type: GlStatus
Privilege: Private

GetExpiration - This operation gets the expiration time and date for a specific file in the distribution pull list.
Arguments: FileName:char *,ExpTimeDate: char*
Return Type: GlStatus
Privilege: Private

ReadNextElement - This operation provides access to the next entry for a file in the distribution "pull" list.
Arguments: RequestInfo:struct
Return Type: GlStatus
Privilege: Private

SetCurrentSize - This operation sets the total size (in KBYTES) of all files in the distribution "pull" list.
Arguments: CurrentSize:unsigned short
Return Type: GlStatus
Privilege: Private

SetExpiration - This operation appropriately sets the expiration date for a file which has been temporarily stored in the distribution "pull" list.
Arguments: FileName:char *,ExpTimeDate:char *
Return Type: GlStatus
Privilege: Private

Associations:

The DsStPullList class has associations with the following classes:
Class: DsStResource isupdatedby
Class: DsStPullMonitor managespullareavia

6.3.15 DsStPullMonitor Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description: To provide a mechanism to delete data objects appropriately from the electronic “pull” data area and to report utilization of the pull area. The Pull Monitor performs these functions by constantly monitoring the contents of the pull area and uses information contained in the list of files contained in the pull area.

Attributes:

$myCacheConfig - This attribute identifies the configuration file for the disk area designated as the "pull" area for electronically distributed files.
Data Type: char *
Privilege: Private
Default Value:

myConfirmTimeOut - This attribute indicates OPS personnel desire to confirm or not to confirm deletion of pulled data files when the files presence in the pull area exceeds the expiration time.
Data Type: boolean
Privilege: Private
Default Value: false

myFreeCache - This attribute indicates the amount of available (i.e., currently unused) disk space (in KBYTES) in the distributed data cache area.
Data Type: long int
Privilege: Private
Default Value:
This is a Derived Attribute.

myHighWaterMark - This attribute indicates the percentage utilization of disk space in th edistributed data cache area which determines nominally when cache reclamation operations start.
Data Type: float

Privilege: Private
Default Value: 75.0

myLowWaterMark - This attribute indicates the percentage amount of disk space usage in the distribution data cache area which determines when cache reclamation operations nominally cease.
Data Type: float
Privilege: Private
Default Value: 45.0

myMaxSize - This attribute indicates the size (in KBYTES) of the disk area designated as the "pull" area.
Data Type: long int
Privilege: Private
Default Value:

myNumberFiles - This attribute indicates the total number of files currently resident in the pull area.
Data Type: long
Privilege: Private
Default Value: 0
This is a Derived Attribute.

mySleepTime - This attribute indicates the time (in seconds) that the Distribution Cache Monitor sleeps before starting cache reclamation (i.e., deleting expired files).
Data Type: float
Privilege: Private
Default Value: 10.0

Operations:

DeleteFile - This operation deletes a file from the designated "pull" area on magnetic disk.
Arguments: FileName: char *
Return Type: GlStatus
Privilege: Public

GetFreeCache - This operation gets the amount of unused disk space in the distributed data cache area.
Arguments: Size:long int
Return Type: GlStatus
Privilege: Private

GetHighWaterMark - This operation gets the high water mark for the pull area (i.e., the percent utilization of the pull area at which the Pull Monitor starts to search for and delete files from the pull area.
Arguments: Percent:float
Return Type: GlStatus
Privilege: Private

GetLowWaterMark - This operation gets the current low water mark for the pull area (i.e., the percent utilization at which the Pull Monitor stops searching for files to delete from the pull area).
Arguments: Percent:float
Return Type: GlStatus
Privilege: Private

GetMaxSize - This operation gets the current total size (in KBYTES) of disk space designated as the "pull" area for electronically distributed files.
Arguments: CacheSize:long int
Return Type: GlStatus
Privilege: Private

GetNumberFiles - This operation gets the total number of files currently in the pull area.
Arguments: TotalFiles:long
Return Type: GlStatus
Privilege: Private

GetSleepTime - This operation gets the time (in seconds) that the Pull Monitor sleeps before checking for and deleting files from the pull area.
Arguments: STime:float
Return Type: GlStatus
Privilege: Private

ReclaimSpace - This operation searches the distributed data cache list for files to delete and subsequently deletes files found.
Arguments:
Return Type: GlStatus
Privilege: Public

ReportStats - This operation reports disk space utilization statistics for the disk space designated as the "pull" area for electronically distributed data files.
Arguments:
Return Type: GlStatus
Privilege: Public

SetFreeCache - This operation sets the amount of unused disk space in the distributed data cache area.

Arguments: Size:long int
Return Type: GlStatus
Privilege: Private

SetHighWaterMark - This operation sets the high water mark for the disk space designated as the "pull" area for electronically distributed files.
Arguments: Percent:float
Return Type: GlStatus
Privilege: Private

SetLowWaterMark - This operation sets the low water mark for the disk space designated as the "pull" area for electronically distributed data files.
Arguments: Percent:float
Return Type: GlStatus
Privilege: Private

SetMaxSize - This operation sets the size (in KBYTES) of the disk space designated as the "pull" area for electronically distributed data files.
Arguments: CacheSize:long int
Return Type: GlStatus
Privilege: Private

SetNumberFiles - This operation sets the number of files that are currently in the pull area.
Arguments: TotalFiles:long
Return Type: GlStatus
Privilege: Private

SetSleepTime - This operation sets the time (in seconds) that the Pull Monitor sleeps before checking for (and deleting when appropriate) files from the "pull" area.
Arguments: STime:float
Return Type: GlStatus
Privilege: Private

Associations:

The DsStPullMonitor class has associations with the following classes:
Class: DsStPullList managespullareavia

6.3.16 DsStRequestManager Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
This class services selected service requests from operations staff and clients. It insures that the service requests are properly routed to the appropriate resource manager(s).

Attributes:

myNumberRequests - This attribute indicates the number of requests for service which have been received but are still in progress.
Data Type: unsigned short
Privilege: Private
Default Value: 0

myRequestTable - This attribute identifies the table of outstanding requests.
Data Type: ptr
Privilege: Private
Default Value:

Operations:

Abort - This public service causes the termination of processing of the specified request if the request is currently queued.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Public

ChangeDeviceStatus - This public service provides operations staff the capability to change the operational status of a specific resource device under control of the Storage Management CI.
Arguments: DeviceName: char *,DeviceStatus: char *
Return Type: GlStatus
Privilege: Public

ChangeRequestPriority - This public service changes the priority of a queued request for service. It has no effect on the request if the request is not currently queued.
Arguments: RequestID:unsigned long,Priority:short
Return Type: GlStatus
Privilege: Public

DismountVolume - This operation dismounts a specific volume.
Arguments: VolName: char *
Return Type: Void
Privilege: Public

EndService - This operation marks the end of service of a request. It removes the entry in the request table which corresponds to the specified request id.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Private

GetDeviceStatus - This public service gets the operational status of a specific resource device.
Arguments: DeviceName: char *,DeviceStatus:char *
Return Type: GlStatus
Privilege: Public

LocateManager - This operation identifies and provides the appropriate resource manager for requests which are still in-progress.
Arguments: RequestID:unsigned long,ResourceManager:ptr
Return Type: GlStatus
Privilege: Private

MountVolume - This operation mounts a specific volume.
Arguments: VolName: char *
Return Type: Void
Privilege: Public

ReportResourceStats - This public service provides various resource related reports for all or specific devices managed by the Storage Management CI.
Arguments: DeviceName: char *
Return Type: GlStatus
Privilege: Public

RequestStatus - This public service requests the status of a previously submitted data store or data retrieve request.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Public

StartService - This operation marks the start of service for a request. It places an entryu in the Request Table which corresponds to the specified request id.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Private

~DsStRequestManager - This is the destructor for the Request Manager object.
Arguments:

Associations:

The DsStRequestManager class has associations with the following classes:
Class: DsStResourceManager routesrequeststo

6.3.17 DsStReservation Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Persistent Class:
Purpose and Description:
This class requests preallocation aand/or reservation of a resource. The Reservation is an internal mechanism to allocate a resource at some future time. The reservation request identifies the requestor, the resource to be allocated and the future time the resource will is required.

Attributes:

myEndTime - This attribute indicates the end time and date for the reservation.
Data Type: char *
Privilege: Public
Default Value:

myPriority - This attribute indicates the priority of the reservation request.
Data Type: unsigned short
Privilege: Public
Default Value: 0

myRequester - This attribute identifes the reservation requester.
Data Type: char *
Privilege: Public
Default Value:

myReservationRef - This attribute identifies the reservation confirmation number for the resource.
Data Type: unsigned long

Privilege: Public
Default Value: 0

myResourceType - This attribute indicates the kind of resource that is being reserved. Possible values include Staging Disk, Network Resource, 4mm Tape, 8mm Tape and CD-ROM.
Data Type: char *
Privilege: Public
Default Value:

mySize - This attribute indicates the amount of data that is anticipated to be transferred by the reserved resource when the reservation becomes active.
Data Type: unsigned short
Privilege: Public
Default Value: 1

myStartTime - This attribute indicates the start time and date of the reservation.
Data Type: char *
Privilege: Public
Default Value:

mySubmitTime - This is the time that the reservation request is submitted.
Data Type: char *
Privilege: Public
Default Value:

Operations:

Activate - This public service activates the resource reservation (i.e., exchanges the resource reservation for a resource allocation).
Arguments: RequestID:unsigned long,ResourceRef:ptr
Return Type: GlStatus
Privilege: Public

Cancel - This public service cancels a previously made reservation for a resource.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Public

Make - This public service creates an initial blank reservation request.
Arguments:
Return Type: GlStatus
Privilege: Public

SetEndTime - This public operation sets the desired end time and date for the resource reservation.
Arguments: TimeDate:char *
Return Type: GlStatus
Privilege: Public

SetPriority - This public service sets the priority of the reservation request.
Arguments: Priority:unsigned:short
Return Type: GlStatus
Privilege: Public

SetRequester - This operation sets the name of the requester of the resource reservation.
Arguments: RequesterName:char
Return Type: GlStatus
Privilege: Public

SetResSize - This public service sets the anticipated size of data (in KBYTES) to be transferred/handled by the resource being reserved.
Arguments: ReservationSize:unsigned short
Return Type: GlStatus
Privilege: Public

SetResourceType - This public service sets the kind of resource that is being reserved.
Arguments: char
Return Type: GlStatus
Privilege: Public

SetStartTime - This operation sets the start time and date for the resource reservation.
Arguments: TimeDate:char *
Return Type: GlStatus
Privilege: Public

Submit - This public service submits the completed reservation request to reserve a resource for some time into the future.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Public

~DsStReservation - This is the destructor for the Reservation object.
Arguments:

Associations:

The DsStReservation class has associations with the following classes:
Class: DsStResourceSchedule isreflectedin

6.3.18 DsStResource Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Persistent Class:
Purpose and Description:
To provide an interface to the services provided by the resources of the data server. The Resource is the generic interface to the specific resources within the Data Server.

Attributes:

myMediaSize - This attribute indicates the size of the media (i.e., small, medium, large) for this allocation.
Data Type: enum
Privilege: Public
Default Value:

myPriority - This attribute indicates the priority of the current allocation of the device.
Data Type: unsigned short
Privilege: Public
Default Value: 0

myResourceName - This attribute uniquely identifies the specific device in the resource pool.
Data Type: char *
Privilege: Public
Default Value:

myStatus - This attribute gives the current status of the resource. Valid values are on-line, off-line, reserved, periodic maintainance, remedial maintenance.
Data Type: char *
Privilege: Public
Default Value:

Operations:

Allocate -

Arguments: ResourceType: char *,MediaType:enum,RequestID: unsigned long,Priority:short, Size:long,ProfileInfo: ptr
Return Type: ResourceRef: ptr
Privilege: Public
PDL:// Log allocation request received
// Project a device use time for the allocation
// If there are no available devices
// Queue request in resource queue
// else
// Check the resource schedule for an available device in the projected time period
// If such a device is found
// Place an entry in the schedule for this device allocation
// Decrement and set the number of available devices
// else
// Queue request in the resource queue
//
// return

Deallocate - This public service releases or frees a previously allocated resource.
Arguments: ResourceRef:ptr
Return Type: GlStatus
Privilege: Public

DistFrom - This public service supports the transfer of data files from internal storage to specific distribution devices.
Arguments: SourceResource:char *,Operation:char *,DataItemName:char *
Return Type: GlStatus
Privilege: Public

GetDevTime - This public service gets the projected device use time for the requested allocation of a resource device.
Arguments: Size:ulong,MediaSize:enum,DeviceTime:unsigned short
Return Type: GlStatus
Privilege: Private

GetMediaSize - This public service provides the size of the media for the allocated resource device.
Arguments: MediaSize:enum
Return Type: GlStatus
Privilege: Public

GetPriority - This operation gets the current priority of the resource.
Arguments: RequestID:unsigned long,CurrentPriority:unsigned short
Return Type: GlStatus
Privilege: Private

GetResourceID - This public service provides the name of the resource allocated.
Arguments: ResourceName:char *
Return Type: GlStatus
Privilege: Public

GetStatus - This public service gets the status of a device in the resource pool.
Arguments: DeviceStatus:char *
Return Type: GlStatus
Privilege: Public

IngestFrom - This public service transfers data files from specific ingest devices to an internal ECS storage device.
Arguments: SourceMachine: char *,SourceDirectory: char *,DestMachine: char *, DestDirectory: char *
Return Type: GlStatus
Privilege: Public

RestoreResourceInformation - This service will restore configurable parameters to preset values. This will be used in restart or cold start situations.
Arguments:
Return Type: GlStatus
Privilege: Private

SetMediaSize - This operation sets the size of the media for the current allocation of the resource device.
Arguments: MediaSize:enum
Return Type: GlStatus
Privilege: Private

SetPriority - This operation sets the current priority of the resource to a specified value. The initial priority is set to the scheduling priority of the request.
Arguments: RequestID:int,NewPriority:short iint
Return Type: GlStatus
Privilege: Private

SetResourceID - This operation sets the name of the resource allocated.
Arguments: ResourceName:char*
Return Type: GlStatus
Privilege: Private

SetStatus - This operation sets the status of a specific device of the resource.
Arguments: DeviceStatus:char *
Return Type: GlStatus
Privilege: Private

~DsStResource - This is the destructor for the resource object.
Arguments:
Return Type: void
Privilege: Public

Associations:

The DsStResource class has associations with the following classes:
Class: DsStPullList isupdatedby
Class: DsStStagingDataList isupdatedby
Class: DsStResourceManager managesaccessto
Class: DsStResourceSchedule providesschedulinginformationto
Class: DsStResourcePolicy utilizes

6.3.19 DsStResourceConfig Class

Parent Class: DsCnConfiguration
Public: NoDistributed Object: No
Purpose and Description:
This class is a specialization of the DsCnConfiguration base class. It manages configuration parameters for resources managed by the Storage Management CI.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsStResourceConfig class has associations with the following classes:
None

6.3.20 DsStResourceManager Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To manage and monitor the resources of the data server. The ResourceManager allocates resources that it controls during the processing of requests.

Attributes:

myFreeResources - This attribute indicates the current number of available devices (i.e., devices which are operational and can be allocated or reserved) in a resource pool.
Data Type: unsigned short
Privilege: Private
Default Value: 0
This is a Derived Attribute.

myMaxResources - This attribute indicates the maximum number of devices in the resource pool. In the case of staging disk it indicates the amount (in KBYTES) of disk space available for workarea allocations and/or reservations.
Data Type: unsigned short
Privilege: Private
Default Value: 0

myReservedResources - This attribute indicates the current number of reservations for devices in the resource pool. In the case of staging disk, it indicates the total current amount of disk space (in KBYTES) for which there is a reservation.
Data Type: unsigned short
Privilege: Private
Default Value: 0
This is a Derived Attribute.

myResourceType - This attribute indicates the name of the resource pool.
Data Type: char *
Privilege: Private
Default Value:

Operations:

AbortRequest - This service causes the abort and/or cancellation of a resource request which has been queued. This service will not cause an abort of a request which is currently active.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Private

AllocateResource - This service allocates a device from a resource pool.
Arguments: MediaType:enum,RequestID:unsigned long,Priority:short,Size:short, ProfileInfo:ptr
Return Type: ResourceRef: ptr
Privilege: Private

ChangeDeviceStatus - This operation enables operations personnel to change the operational status of a specific resource device.
Arguments: RequestID:unsigned long,NewStatus:short int
Return Type: GlStatus
Privilege: Private

ChangePriority - This public service enables the changing of a resource request's priority. This service will only change the priority of a request which is queued.
Arguments: RequestID:unsigned long,Priority:short int
Return Type: GlStatus
Privilege: Private
PDL:// Log change priority request received

```
//  Search the resource queue for a specific requestID
//  If the request was found in the queue
//      Set the new priority for the request
//      Notify requester that priority was changed
//      Log priority changed for request
//  else
//      Search the resource schedule to ascertain if this is an active request
//      If the request was found in the schedule
//          Notify requester that priority is not changed for active requests
//      else
//          Notify requester that priority is not changed for requests which have completed
//
//  Log change priority request completed
//  return
```

DeallocateResource - This operation deallocates a previously allocated device in the resource pool.
Arguments: ResourceRef:ptr

Return Type: void
Privilege: Private

DismountVolume - This operation enables operations personnel to dismount a previously mounted volume.
Arguments: VolName:char *
Return Type: GlStatus
Privilege: Private

GetFreeResources - This operation gets the current number of free resources in the resource pool.
Arguments: NumResources:short int
Return Type: GlStatus
Privilege: Private

GetReservedResources - This operation gets the current number of reservations for devices in the resource pool.
Arguments: NumResources:short int
Return Type: GlStatus
Privilege: Private

MountVolume - This operation enables operations personnel to mount a specific volume.
Arguments: VolName:char *
Return Type: GlStatus
Privilege: Private

ReportResourceInformation - This service reports information concerning devices in the resource pool.
Arguments:
Return Type: GlStatus
Privilege: Private

RequestStatus - This operation provides a response to queries received requesting the status of previously submitted store and retrieval requests.
Arguments: RequestID:unsigned long
Return Type: GlStatus
Privilege: Private

SetFreeResources - This operation sets the current number of available (i.e, unallocated) devices.
Arguments: NumResources:short int
Return Type: GlStatus
Privilege: Private

Associations:

The DsStResourceManager class has associations with the following classes:
Class: DsStResource managesaccessto
Class: DsStResourceQueue retainsrequestsqueuedby
Class: DsStRequestManager routesrequeststo
Class: DsStResourceSchedule schedulesresourcesvia

6.3.21 DsStResourcePolicy Class

Parent Class: Not Applicable
Public: No Distributed Object: No
Purpose and Description: This class maintains information concerning the operations staff desired policy(ies) for scheduling resources. A Device Table created and maintained by this class provides quick access to this information.

Attributes:

$myDeviceTable - This attribute identifies the table of projected device use times for each type of resource (excepting those resources which are not allocated or reserved).
Data Type: ptr
Privilege: Private
Default Value:

$myPolicyConfig - This attribute identifies persistent policy configuration information for each type of resource.
Data Type: ptr
Privilege: Private
Default Value:

myDismountTime - This attribute specifies the time in seconds required by OPS personnel to dismount a single piece of media from a device in the resource pool.
Data Type: float
Privilege: Private
Default Value: 30.0

myEfficiencyFactor - This attribute specifies the efficiency factor reflecting how well and how proficient OPS personnel are when handling media during mount and dismount operations.
Data Type: float
Privilege: Private
Default Value: 1.0

myHighDensitySize - This attribute specifies the maximum size (in KBYTES) that can be stored on a single piece of media which is considered to be "Large" in size.
Data Type: int
Privilege: Private
Default Value: 10000

myLowDensitySize - This attribute indicates the maximum size of data (in KBYTES) that can be stored on a single piece of media which is considered to be "Small" in size.
Data Type: int
Privilege: Private
Default Value: 500

myMaxNumberMedia - This attribute indicates the maximum number of media that are allowed for a resource for a single distribution or ingest request.
Data Type: int
Privilege: Private
Default Value: 10

myMedDensitySize - This attribute specifies the maximum size (in KBYTES) of data that can be stored on a single piece of media which is considered to be "Medium" in size.
Data Type: int
Privilege: Private
Default Value: 2000

myMediaTypes - This attribute indicates the number of different densities of media that can be read/written by a device in the resource pool.
Data Type: int
Privilege: Private
Default Value: 3

myMountTime - This attribute specifies the time required in seconds for OPS personnel to mount one piece of media on a device of the resource pool.
Data Type: float
Privilege: Private
Default Value: 30.0

myNumberDevices - This attribute specifes the number of physical devices of this type of resource.
Data Type: unsigned short
Privilege: Private
Default Value: 1

myRewindTime - This operation indicates the time (in seconds) to rewind a single piece of media on a device in the resource pool.
Data Type: float

Privilege: Private
Default Value: 60.0

myTransferRate - This attribute specifies the transfer rate in KBYTES per second of a device in the resource pool.
Data Type: float
Privilege: Private
Default Value: 0.0

Operations:

BuildDevTable - This operation constructs a table of values used to project the total time of use (i.e., from the time the first piece of media for the device is mounted until the time the last piece of media is unmounted) for a resource for which an allocation request has been received.
Arguments:
Return Type: GlStatus
Privilege: Private

GetNumberDevices - This operation provides access to the number of devices in a specific resource pool.
Arguments: Devices:short
Return Type: GlStatus
Privilege: Private

ProjectDuration - This operation provides an estimated time of use of a device. The estimate is based upon the specified time to mount and dismount all media involved in the request and the actual transfer time of data.
Arguments: Size:long,MediaType:enum,Duration:short

SetDismountTime - This operation set the time (in seconds) operations personnel take to dismount a single piece of media from a device in the resource pool.
Arguments: real
Return Type: GlStatus
Privilege: Private

SetMountTime - This operation sets the time (in seconds) operations personnel take to mount a single piece of media on a device in a specific resource pool.
Arguments: MountTime:float
Return Type: GlStatus
Privilege: Private

SetNumberDevices - This operation sets the number of devices in the resource pool.
Arguments: unsigned short
Return Type: GlStatus
Privilege: Private

SetTransferRate - This operation sets the data transfer rate (in KBYTES/second) of a device in the resource pool.
Arguments: Rate:float
Return Type: GlStatus
Privilege: Private

Associations:

The DsStResourcePolicy class has associations with the following classes:
Class: DsStResource utilizes

6.3.22 DsStResourceQueue Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This class maintains a list of pending resource requests. Outstanding requests for allocation of resources are queued based upon their respective priority. These requests are dequeued based upon this same priority and resource availability. Each Resource Manager maintains a unique queue for the resources it manages.

Attributes:

myCurrentqueued - This attribute indicates the current number of entries in the resource queue.
Data Type: unsigned short
Privilege: Private
Default Value: 0
This is a Derived Attribute.

myMaxSize - This is the maximum size of the Resource Queue.
Data Type: unsigned short
Privilege: Private
Default Value:

Operations:

Dequeue - This service will remove an entry from the queue
Arguments: RequestInfo: structure
Return Type: GlStatus
Privilege: Private

FindRequest - This operation locates a specific request in the resource queue.
Arguments: RequestID:unsigned long, RequestInfo: structure
Return Type: GlStatus
Privilege: Private

GetCurrentQueued - This operation indicates the number of requests currently in the resource queue.
Arguments: NumRequests: short
Return Type: GlStatus
Privilege: Private

GetNextQueued - This operation provides information about the next element in the resource queue. Elements are queued by high priority.
Arguments: RequestInfo: structure
Return Type: GlStatus
Privilege: Private

Queue - This service will add an item to the queue
Arguments: RequestInfo: structure
Return Type: GlStatus
Privilege: Private

RestoreQueue - This service will restore queue information to a previously checkpointed condition.
Arguments:
Return Type: GlStatus
Privilege: Private

SetPriority - This operation sets the priority of a specific request in the resource queue.
Arguments: Priority: unsigned short,RequestID: unsigned long
Return Type: GlStatus
Privilege: Private

Update - This service will update information about an element in the queue.
Arguments: RequestInfo: structure

Return Type: GlStatus
Privilege: Private

Associations:

The DsStResourceQueue class has associations with the following classes:
Class: DsStResourceManager retainsrequestsqueuedby

6.3.23 DsStResourceSchedule Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To schedule the usage of resources of the data server. The ResourceSchedule is a compilation of all future resource requirements.

Attributes:

$mySchedulingConfig
Data Type: char *
Privilege: Private
Default Value:

myEndDate - This attribute indicates the end time and date of the current view of the schedule. This time and date are changed on a daily basis.
Data Type: char *
Privilege: Private
Default Value:
This is a Derived Attribute.

myPoolName - This attribute indicates the name of the resource pool (e.g., Staging disk, CDROM, Archive, Tape).
Data Type: char *
Privilege: Private
Default Value:

myScheduleFileName - This is the external filename of the resource schedule for all devices in the resource pool.
Data Type: char *

Privilege: Private
Default Value:

myStartDate - This attribute indicates the start time and date of the current view of the resource schedule. This time and date are changed appropriately on a daily basis.
Data Type: char *
Privilege: Private
Default Value:
This is a Derived Attribute.

Operations:

ActivateReservation - This operation confirms a resource reservation and provides a unique reference to the reserved resource. It in effect, exchanges a reservation for a resource for the actual resource object.
Arguments:
Return Type: GlStatus
Privilege: Private

BookResource - This operation makes an entry in the resource schedule for each allocation made of a device in the resource pool.
Arguments: ResourceRef:unsigned long,RequestID:unsigned long,StartTime:char *,EndTime:char* Size:unsigned short,MediaType:enum,Priority:unsigned short, ProfileINfo:ptr,Type:enum
Return Type: GlStatus
Privilege: Private

CancelReservation - This public service cancels a previously made reservation for a resource. Before cancelling a reservation, this service confirms that this is a valid request (i.e., only the requester may cancel the reservation).
Arguments: ReservationRef:unsigned long,RequestID:unsigned long
Return Type: GlStatus
Privilege: Private

CheckReservations
Arguments: CheckQueue: boolean
Return Type: GlStatus
Privilege: Private

CreateSchedule - This operation creates the initial schedule for all devices in a resource pool.
Arguments:

Return Type: GlStatus
Privilege: Private

FindRequest - This operation searches the current view of the resource schedule for a specific in-progress request. If the request is found then information about that request and its resource usage is provided.
Arguments: RequestID:unsigned long,RequestInfo:struct
Return Type: GlStatus
Privilege: Private

FindResource - This operation searches the resource schedule for the first device which is available (i.e., that is not currently allocated or reserved) in a designated timeframe. If such a device is found, then a unique reference to the resource is provided.
Arguments: StartTime:char *,EndTime:char *,ResourceRef:unsigned long
Return Type: GlStatus
Privilege: Private

GetCurrentTime - This operation gets the current system time and puts it into the needed format.
Arguments: CurrentTime:char *
Return Type: GlStatus
Privilege: Private

RecordEndTime - This operation records into the resource schedule the actual time at which a resource is freed and released for subsequent use by another requester.
Arguments: ResourceRef:unsigned long,CompletionStatus:long
Return Type: GlStatus
Privilege:

ReportSchedule - This operation reports scheduling information about resources in the current view of the resource schedule.
Arguments: DeviceUse: Enum,StartTime:char *,EndTime:char *
Return Type: GlStatus
Privilege: Private

RestoreSchedule - This operation restores a previously checkpointed resource schedule.
Arguments:
Return Type: GlStatus
Privilege: Private

RollSchedule - This operation sets the current view of the resource schedule. The current view of the schedule is defined by the start time and end time attributes.
Arguments: NumberDays:unsigned short
Return Type: GlStatus
Privilege: Private

SaveSchedule - This service will checkpoint a resource schedule.
Arguments:
Return Type: GlStatus
Privilege: Private

Associations:

The DsStResourceSchedule class has associations with the following classes:
Class: DsStReservation isreflectedin
Class: DsStResource providesschedulinginformationto
Class: DsStResourceManager schedulesresourcesvia

6.3.24 DsStSCacheConfig Class

Parent Class: DsCnConfiguration
Public: NoDistributed Object: No
Purpose and Description:
This class is a specialization of the DsCnConfiguration base class. It manages configuration parameters for the staging disk cache area.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsStSCacheConfig class has associations with the following classes:
None

6.3.25 DsStSchedulingConfig Class

Parent Class: DsCnConfiguration
Public: NoDistributed Object: No
Purpose and Description:
This class is a specialization of the DsCnConfiguration base class. It manages configuration parameters for scheduling resources managed by the Storage Management CI. Resources currently managed by the CI include 4mm Tape, 8mm Tape, CDROM, Staging Disk, the ECS Archive and the ECS communications network.

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsStSchedulingConfig class has associations with the following classes:
None

6.3.26 DsStStagingDataList Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
This class provides an mechanism for tracking the contents of non-work areas of staging disk used by the Data Server. The dynamic record of the data objects currently on staging disk allows the Data Server to cache data files on staging disk and subsequently provide quicker retrieval access to those data files.

Attributes:

myCurrentSize - This attribute indicates the current number of files resident in staging disk cache.
Data Type: short int
Privilege: Private

Default Value: 0
This is a Derived Attribute.

myNextElement - This attribute indicates the next file on the staging data list.
Data Type: ptr
Privilege: Private
Default Value:
This is a Derived Attribute.

myTotalSize - This attribute indicates the total size (in KBYTES) of all files currently in staging disk cache.
Data Type: long int
Privilege: Private
Default Value: 0

Operations:

AddElement - This operation adds a file to the list of files currently in the staging disk cache area. In addition to the file name, other information retained in the list includes: RequestID, file size, priority, reference count and the time and date of insertion in the cache.
Arguments: FileName:char *,RequestID:unsigned long,Size:long,Priority:short
Return Type: GlStatus
Privilege: Private

AssociateElement - This operation searches the staging disk cache list for a specific file. If the file is found, the reference count for the file is incremented by 1 to reflect an additional request for the file.
Arguments: FileName:char *
Return Type: GlStatus
Privilege: Private

DeleteElement - This operation removes a file from the staging disk cache list.
Arguments: FileName:char *,RequestID:long
Return Type: GlStatus
Privilege: Private

DisassociateElement - This operation decrements the reference count for a specified file by one. This decrementing indicates that the need for the file by the requester has been satisfied.
Arguments: FileName:char *
Return Type: GlStatus
Privilege: Private

GetCurrentSize - This operation gets the total size (in KBYTES) of all files currently in the staging disk cache area.
Arguments: TotalFiles:short int
Return Type: GlStatus
Privilege: Private

GetTotalSize - This operation gets the total size (in KBYTES) of all files currently in staging disk cache.
Arguments: TotalSize:long int
Return Type: GlStatus
Privilege: Public

ReadNextElement - This operation provides access to the next file in the staging data cache list. This operation will be used while searching the staging data cache list for files to delete.
Arguments: NextFile:ptr
Return Type: GlStatus
Privilege: Private

SetCurrentSize - This operation sets the total size (in KBYTES) of all files currently in the staging disk cache area.
Arguments: TotalFiles:short int
Return Type: GlStatus
Privilege: Private

SetExpiration - This operation sets the file expiration time and date. At this time the file becomes eligible for deletion from staging disk cache.
Arguments: FileName:char *,TimeDate:char *
Return Type: GlStatus
Privilege: Private

SetTotalSize - This operation sets the total size (in KBYTES) of all files currently in staging disk cache.
Arguments: TotalSize:long int
Return Type: GlStatus
Privilege: Public

Associations:

The DsStStagingDataList class has associations with the following classes:
Class: DsStResource isupdatedby
Class: DsStStagingMonitor managesstageddatavia

6.3.27 DsStStagingDisk Class

Parent Class: DsStStorageResource
Public: YesDistributed Object: Yes
Persistent Class:
Purpose and Description:
To provide an interface to the staging disk resource for the temporary storage of data. The StagingDisk provides temporary and buffer storage.

Attributes:

myAvailableSpace - This attribute indicates the amount of free disk space (in KBYTES) remaining for usage.
Data Type: long int
Privilege: Public
Default Value: 0
This is a Derived Attribute.

myMaxSize - This attribute indicates the maximum size (in KBYTES) of staging disk that can be allocated to the user at any one time.
Data Type: long int
Privilege: Private
Default Value:

myPathname - This is the pathname to the directory associated with this staging disk.
Data Type: char *
Privilege: Public
Default Value:

mySize - This attribute indicates the size (in KBYTES) of the staging disk allocated.
Data Type: long int
Privilege: Public
Default Value: 0

Operations:

CopyFile - This operation copies a file on staging disk.
Arguments: SourceFile:char *,DestFile:char *
Return Type: GlStatus
Privilege: Private

ExtendStaging - This operation extends the size of an allocation of staging disk up to but not exceeding a hard limit. This hard limit is predetermined and set by operations personnel.
Arguments: ExtendSize:short
Return Type: GlStatus
Privilege: Private

GetAvailableSpace - This operation gets the amount of available disk space for an allocation.
Arguments: AvailableSpace:long
Return Type: GlStatus
Privilege: Public

GetMaxSize - This operation gets the maximum size of the staging disk allocation.
Arguments: MaxSize:long
Return Type: GlStatus
Privilege: Public

GetPathName - This operation gets the path name for the allocation of staging disk.
Arguments: PathName:char *
Return Type: GlStatus
Privilege: Private

GetSize - This operation gets the current size of the staging disk allocation.
Arguments: CurrentSize:long
Return Type: GlStatus
Privilege: Public

SetAvailableSpace - This operation sets the amount of available disk space for the allocation.
Arguments: AvailableSpace:long
Return Type: GlStatus
Privilege: Private

SetMaxSize - This operation sets the maximum size for the allocation of staging disk.
Arguments: Size:long

Return Type: GlStatus
Privilege: Private

SetPathName - This operation sets the path name for the allocation of staging disk.
Arguments: PathName:char *
Return Type: GlStatus
Privilege: Public

SetSize - This operation sets the current size of the staging disk allocation.
Arguments: CurrentSize:long
Return Type: GlStatus
Privilege: Private

~DsStStagingDisk - This operation is the destructor for the StagingDisk object.
Arguments:
Return Type: void
Privilege: Public

Associations:

The DsStStagingDisk class has associations with the following classes:
None

6.3.28 DsStStagingFile Class

Parent Class: DsStFile
Public: YesDistributed Object: Yes
Purpose and Description:
This class provides access to file level services for files created in allocated workareas on staging disk. This class is a generalization of the superclass File.

Attributes:

All Attributes inherited from parent class

Operations:

DeleteFile - This operation deletes the specified file from a workarea on staging disk.
Arguments: FileName: char *
Return Type: GlStatus

Privilege: Public
PDL:// Get the size of the file being deleted
// Delete the file from staging disk
// Decrement the size of the file from the running total of bytes associated with
// this allocation of staging disk
// Decrement the size of the file from the
// running total of bytes associated with the directory on staging disk

Put - This public service provides the capability to write data files to a workarea on staging disk. It also keeps track of the total amount of disk used.
Arguments: FileName: char *
Return Type: GlStatus
Privilege: Public
PDL: // Get file size
// Add file size to running total for allocation
// If running total exceeds hard limit for disk
// usage
// Log hard limit exceeded
// Report error to OPS personnel
// else
// If running total exceeds soft limit for
// disk usage
// Log soft limit exceeded
// Increase soft limit by extension size
// Extend allocation to soft limit
// Add file size to running total for the directory
// Continue Put file operation
// .
// .
// return

~DsStStagingFile - This is the destructor for the StagingFile object.
Arguments:

Associations:

The DsStStagingFile class has associations with the following classes:
None

6.3.29 DsStStagingMonitor Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To privde a mechanism to delete data objects appropriately from non-work areas of staging disk and to report utilization of staging disk. The StagingMonitor performs these functions by constantly monitoring the contents of staging disk. Via information contained in theStagingDataList, it determines the correct time to delete files which have been placed on staging disk. It also deletes other data objects which have been distributed via electronic push and/or physical media.

Attributes:

$myCacheConfig - This attribute identifies the configuration file for the staging disk cache.
Data Type: ptr
Privilege: Private
Default Value:

myFreeCache - This attribute indicates the total size of the staging disk cache area.
Data Type: long int
Privilege: Private
Default Value:

myHighWaterMark - This attribute indicates the percentage utilization of the staging disk cache area which nominally determines the start of space reclamation operations.
Data Type: real
Privilege: Private
Default Value: 75

myLowWaterMark - This attribute indicates the percentage utilization of the staging disk cache which nominally determines when space reclamation operations cease.
Data Type: real
Privilege: Private
Default Value: 40

myMaxSize - This attribute indicates the total size (in KBYTES) of the staging disk cache.
Data Type: long int
Privilege: Private
Default Value:

mySleepTime - This attribute indicates the time in seconds that the staging disk monitor will sleep between the start of space reclamation operations.

Data Type: real
Privilege: Private
Default Value: 10

Operations:

Delete - This operation deletes a specified file from the staging disk cache area.
Arguments: Filename:char
Return Type: GlStatus
Privilege: Private

GetFreeCache - This operation gets the size of staging disk which is currently free and not in use.
Arguments: Size:long int
Return Type: GlStatus
Privilege: Private

GetHighWaterMark - This operation gets the high water mark (percent utilization) for the staging disk cache area.
Arguments: Percent:real
Return Type: GlStatus
Privilege: Private

GetLowWaterMark - This operation gets the low water mark (percent utilization) for the staging disk cache area.
Arguments: Percent:real
Return Type: GlStatus
Privilege: Private

GetMaxSize - This operation gets the maximum size of disk space in the staging disk cache.
Arguments: Size: long int
Return Type: GlStatus
Privilege: Private

GetSleepTime - This operation gets the operations determined sleep time for the Staging Monitor.
Arguments: Time:real
Return Type: GlStatus
Privilege: Private

ReclaimSpace - This operation searches the list of files currently resident in the staging disk cache for files which can be deleted. Files found as a result of this operation are

deleted.
Arguments:
Return Type: GlStatus
Privilege: Private

ReportStats - This operation provides reports on staging disk cache statistics.
Arguments:
Return Type: GlStatus
Privilege: Public

SetFreeCache - This operation sets the size of disk space currently available (i.e., unused) in staging disk cache.
Arguments: Size:long int
Return Type: GlStatus
Privilege: Private

SetHighWaterMark - This operation sets the high water mark (percent utilization) for staging disk cache.
Arguments: Percent:real
Return Type: GlStatus
Privilege: Private

SetLowWaterMark - This operation sets the low water mark (percent utilization) for the staging disk cache area.
Arguments: Percent:real
Return Type: GlStatus
Privilege: Private

SetMaxSize - This operation sets the maximum size of staging disk cache (in KBYTES).
Arguments: Size: long int
Return Type: GlStatus
Privilege: Private

SetSleepTime - This operation sets the sleep time of the Staging Monitor. The Staging Monitor periodically awakens from its hibernation state to check for and to subsequently delete files on staging disk.
Arguments: Time:real
Return Type: GlStatus
Privilege: Private

Associations:

The DsStStagingMonitor class has associations with the following classes:
Class: DsStStagingDataList managesstageddatavia

6.3.30 DsStStorageResource Class

Parent Class: DsStResource
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To provide an interface to the services provided by the set of storage resources utilized by the DataServer. The StorageResource is a type of resource that is used for providing persistent storage devices for the DataServer.

Attributes:

All Attributes inherited from parent class

Operations:

Backup - This service will perform a backup of a storage resource.
Arguments:
Return Type: GlStatus
Privilege: Private

Restore - This service will restore the data files on a storage resource to its previous state using a previously made backup of the storage resource.
Arguments:
Return Type: GlStatus
Privilege: Private

Associations:

The DsStStorageResource class has associations with the following classes:
None

6.3.31 DsStTape Class

Parent Class: DsStPhysicalResource

Public: YesDistributed Object: Yes

Purpose and Description:

This class provides an interface to tape resources managed by Storage Management. The Ingest Client to Storage Management can use tape resources to ingest data files. The Data Distribution Client to Storage Management can use tape resources to distribute data files to requesting users.

Attributes:

myBlockCount - This attribute indicates the current number of blocks which have been read from or written to the media currently mounted on the tape device.
Data Type: long
Privilege: Public
Default Value: 0

myBlockSize - This attribute indicates the block size (in bytes) used for reading and writing to tape.
Data Type: long
Privilege: Private
Default Value: 0

myCapacity - This attribute indicates the capacity (in KBYTES) of the media currently mounted in the tape resource.
Data Type: long
Privilege:
Default Value: 0

myNextBlock - This attribute indicates the next block on the media which can be read or written to.
Data Type: long
Privilege: Private
Default Value: 0

myRemainingBlocks - This attribute indicates the number of blocks remaining on the media.
Data Type: long
Privilege: Public
Default Value: 0

Operations:

Dismount - This public operation dismounts the specified volume from a tape device.
Arguments: VolName: char *
Return Type: GlStatus
Privilege: Public

GetBlockCount - This operation gets the current number of blocks which have been read or written to the media currently mounted on the tape device.
Arguments: BlockCount: long
Return Type: GlStatus
Privilege: Public

GetBlockSize - This operation gets the blocksize (in bytes) used for read and write operations to the tape device.
Arguments: BlockSize: long
Return Type: GlStatus
Privilege: Public

GetNextBlock - This operation gets the next block to be read or written from the volume currently mounted on the tape resource.
Arguments: NextBlock: long
Return Type: GlStatus
Privilege: Public

GetRemainingBlocks - This public service gets the number of blocks remaining on the tape media which can be used to record data files.
Arguments: AvailBlocks: long
Return Type: GlStatus
Privilege: Public

Mount - This public service mounts the specified volume onto a tape device.
Arguments: VolName: char *
Return Type: GlStatus
Privilege: Public

Rewind - This public service rewinds the media currently mounted on the tape device.
Arguments:
Return Type: GlStatus
Privilege: Public

SetBlockCount - This operation sets the number of blocks which have been currently read or written from or to the media currently mounted.
Arguments: BlockCount: long

Return Type: GlStatus
Privilege: Private

SetBlockSize - This operation sets the block size (in bytes) for read and write operations to the tape resource.
Arguments: BlockSize: long
Return Type: GlStatus
Privilege: Private

SetNextBlock - This operation sets the next block to be read or written from the volume currently mounted on the tape resource.
Arguments: NextBlock: long
Return Type: GlStatus
Privilege: Private

SetRemainingBlocks - This operation sets the number of blocks remaining which have been read from or written to the tape media.
Arguments: AvailBlocks: long
Return Type: GlStatus
Privilege: Private

Associations:

The DsStTape class has associations with the following classes:
None

6.3.32 GlLog Class

Parent Class: Not Applicable

Attributes:

None

Operations:

None

Associations:

The GlLog class has associations with the following classes:
None

6.3.33 GlNotification Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To alert or inform clients of events. The Notification transmits data from a notification source (e.g. Subscription that has been triggered) to a notification receiver (e.g. a user to created a subscription). It attempts to notify the recipient directly (via the DsClNotificationReceiver object), else via email.

Attributes:

None

Operations:

None

Associations:

The GlNotification class has associations with the following classes:
None

6.4 CSCI Dynamic Model

This section describes several scenarios for the Storage Management CSCI. A scenario delineates the steps that occur in a particular execution through the system.

The scenarios address the following topics:

Resources

- Allocation of a Physical Resource
- Deallocation of a Physical Resource
- Deletion of Staging Disk Data
- Deletion of Pull Disk Data With Confirmation by Operations Personnel

- Setting the state of a Tape Device to Off-line
- Submitting a Resource Reservation
- Activating a Resource Reservation
- Canceling a Resource Reservation

Requests

- Aborting a request
- Insert Data Requests
- Retrieve Data Requests

6.4.1 Aborting a Request for Service

Summary

The calling object issues a request to abort a previous request. The abort request is received and logged. The request to be aborted is located in the resource queue and subsequently removed. The abort is logged and a notification of abnormal termination is sent to the requester.

Assumptions

This scenario assumes that there are no errors.

6.4.2 Activating a Resource Reservation

Summary

The calling object submits a request to allocate the resource it has previously reserved. The request is received and the reservation for the resource is confirmed by examining the resource schedule. Appropriate counters are subsequently updated and the calling object receives a distributed object indicated by the resource reference.

Assumptions

This scenario assumes that there are no errors.

6.4.3 Allocation of a Physical Resource, No Resource Available

Summary

The calling object indicates the type of resource desired in the allocate request. The allocation request is received and based upon input parameters in the request a projected end time for the allocation is made. No device of the desired resource type is found which satisfies the time constraints for the request. The request is queued.

Assumptions

This scenario assumes that there are no errors. It also assumes that all devices of the desired resource type are unavailable (i.e., they are all currently allocated and/or reserved for some other client for a portion of the time period projected for the allocation request).

6.4.4 Allocation of a Physical Resource, Resource Available

Summary

The calling object indicates the type of resource desired in the allocate request. The allocation request is received and based upon input parameters in the request a projected end time for the allocation is made. The first device of the desired resource type is found which satisfies the time constraints for the request. The schedule is updated to reflect the specific resource allocated. The calling object receives a distributed object indicted by the resource reference.

Assumptions

This scenario assumes that there are no errors. It also assumes that a device of the desired resource type is available for allocation to a client.

6.4.5 Inserting Data into the Archive

Summary

The calling object requests that a specified file be inserted into the archive. The archive object copies the file into the archive and returns a status to the caller.

Assumptions

This scenario assumes that there are no errors.

6.4.6 Retrieving Data from the Archive, Checksum Error

Summary

The calling object requests that a specified file be retrieved from the archive. The archive object checks to see if the requested file is already located on the staging disk. If the file is not already present, the file is retrieved from the archive. After retrieval of the file, a checksum is performed and the results are compared to the checksum performed when the file was originally archived. After the checksum error is detected proper notifications are sent and the error logged. The file is not placed in the staging disk cache list.

Assumptions

There is only one file to be retrieved indicated in the request. The retrieve operation will succeed. The requested file is not already on the staging disk.

6.4.7 Retrieving Data from the Archive, File not in Cache

Summary

The calling object requests that a specified file be retrieved from the archive. The archive object checks to see if the requested file is already located on the staging disk. If the file is not already present, the file is retrieved from the archive. After retrieval of the file, a checksum is performed and the results are compared to the checksum performed when the file was originally archived. Additionally, the file is now placed in the staging disk cache list.

Assumptions

There is only one file to be retrieved indicated in the request. The retrieve operation will succeed. The requested file is not already on staging disk.

6.4.8 Canceling a Resource Reservation

Summary

The calling object decides to cancel a previously made resource reservation. The cancellation request is received and the reservation reference is verified. Appropriate counters are updated.

Assumptions

This scenario assumes that there are no errors.

6.4.9 Deallocation of a Physical Resource, Queued Requests

Summary

The calling object has completed its use of the physical resource and indicates this completion by deallocating the resource. The deallocation request is received and the actual end time for the use of the resource is recorded in the schedule. To prevent idle resources, the queue of requests for this resource type is searched after determining that there is no imminent reservation for a resource of the type deallocated. The highest priority queued request which can be satisfied is dequeued.

Assumptions

This scenario assumes that there are no errors. It also assumes that there is no reservation for the device just deallocated.

6.4.10 Deallocation of a Physical Resource, Imminent Reservation for Resource

Summary

The calling object has completed its use of the physical resource and indicates this completion by deallocating the resource. The deallocation request is received and the actual end time for the use of the resource is recorded in the schedule. It is determined that there is pending reservation for this device such that there is insufficient time to service any queued requests for this resource. Hence, the resource queue is not searched for a request to satisfy in the available time between now and when the reservation is scheduled to start.

Assumptions

This scenario assumes that there are no errors. It also assumes that there is a queue of requests for the resource type just deallocated.

6.4.11 Deletion of Pull Disk Data with Operations Confirmation

Summary

The Pull Monitor periodically wakes up and checks the need to delete files in the pull disk area. This initial check is to determine if the high water mark (highest desired percent utilization of the pull area) has been reached. If it has been reached or surpassed the Pull Monitor searches through the pull data list for files it can delete. It deletes eligible files found until all files on the list have been checked or the low water mark (lowest desired percent utilization of the cache area) is reached, whichever comes first. Before deleting a file, the Pull Monitor confirms the desire of Operations personnel to actually delete the file. After confirmation is obtained, the Pull Monitor deletes the file. After deleting appropriate files, the Pull Monitor returns to a hibernation state.

Assumptions

This scenario assumes that there are no errors, that Operation has set the “confirm delete flag”, that the high water mark has been surpassed, and that the operator immediately confirms the deletion of the one file that was found eligible for deletion.

6.4.12 Deletion of Staging Disk Data

Summary

The Staging Monitor periodically wakes up and checks the need to delete files in the staging disk cache. This initial check is to determine if the high water mark (highest desired percent utilization of the cache area) has been reached. If it has been reached or surpassed the Staging Monitor searches through the staging data list for files it can delete. It deletes eligible files until all files on the list have been checked or the low water mark (lowest desired percent utilization of the cache area) is reached, whichever comes first. After deleting appropriate files, the Staging Monitor returns to a hibernation state.

Assumptions

This scenario assumes that there are no errors. It also assumes that the high water mark has been surpassed and that only one file is eligible for deletion.

6.4.13 Setting the Operational State of a Tape Device

Summary

Operations personnel have determined that a specific tape device needs to be taken off-line. The operator issues the appropriate request. The request is received and before setting the new operational state, the current operational state is checked to verify that the state change is not illegal or inappropriate.

Assumptions

This scenario assumes that the specific tape device designated to be taken off-line is on-line. The system is quiescent.

6.4.14 Submitting a Resource Reservation

Summary

The resource reservation request is received and based upon input parameters in the request a search of the resource's schedule is performed. The first device of the desired resource type is found which satisfies the time constraints for the reservation request. The schedule is updated to reflect a reservation for that specific device. The calling object receives a reservation reference (confirmation number) to document the reservation for the resource.

Assumptions

This scenario assumes that there are no errors. It also assumes that a device of the desired resource type is available for allocation to a client.

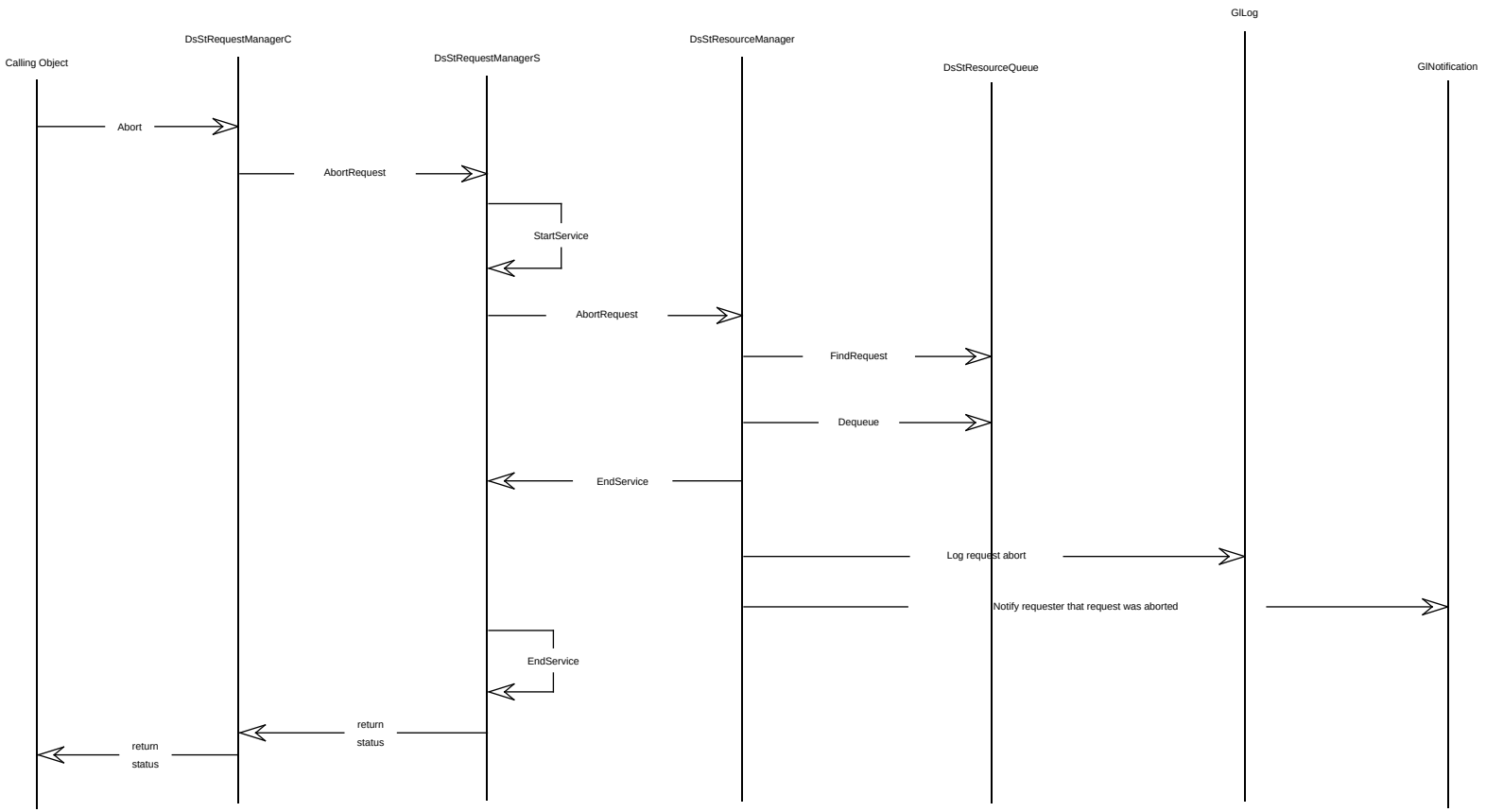
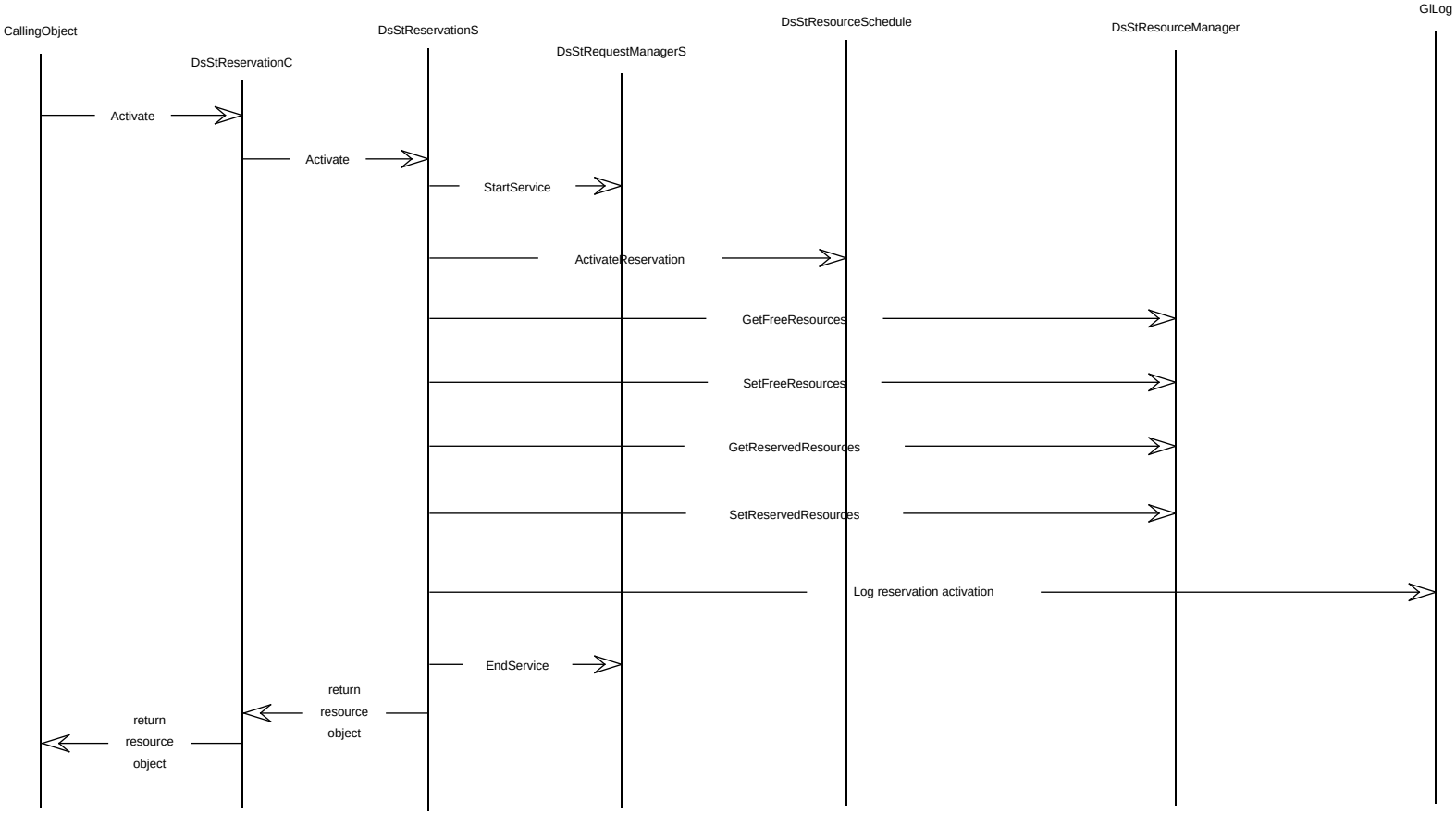


Figure 6.4-1. ABORT_REQUEST Dynamic Model



6.4-2. ACTIVATE_RESERVATION Dynamic Model

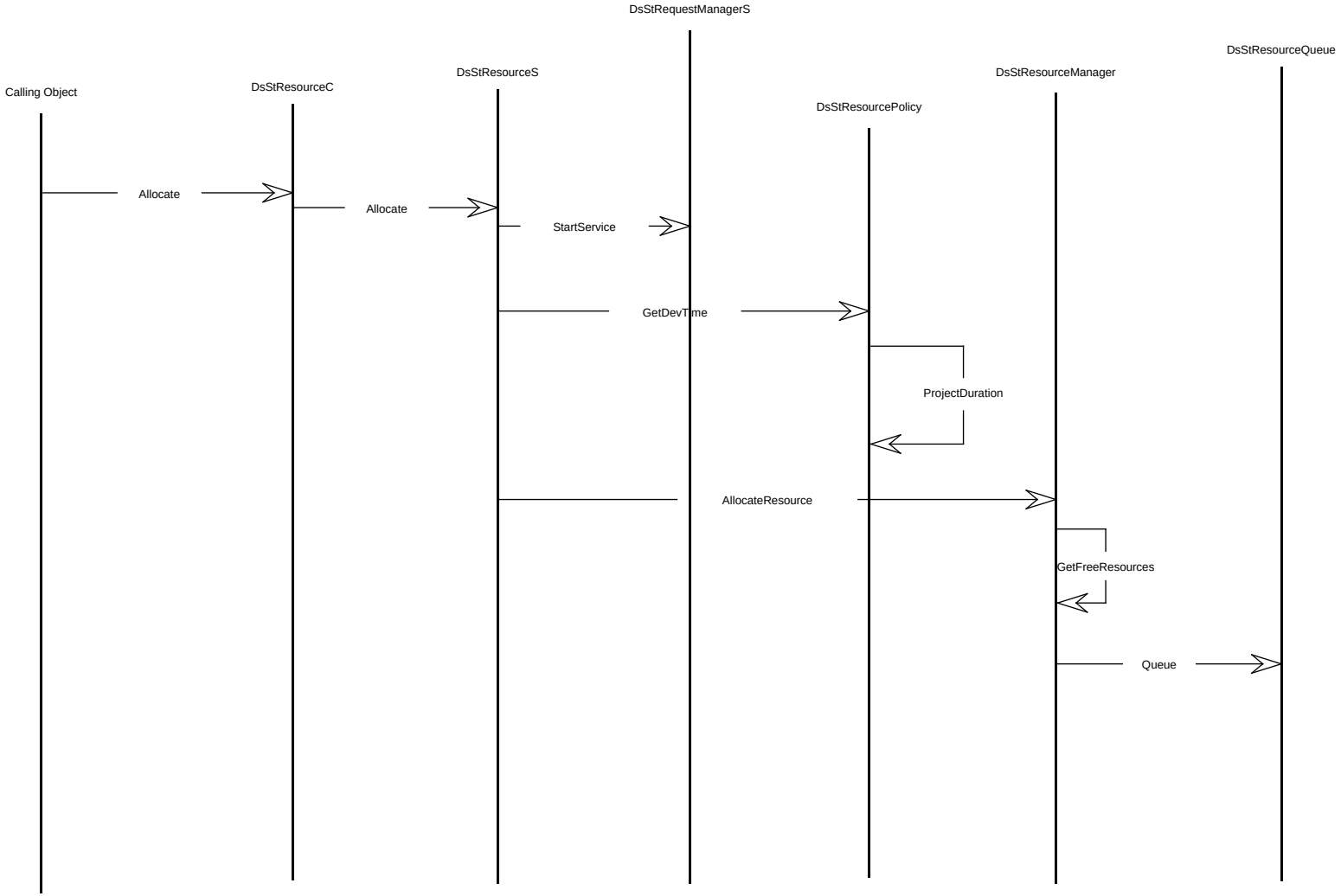


Figure 6.4-3. ALLOCATEPHYSRES_NORESAVAIL Dynamic Model

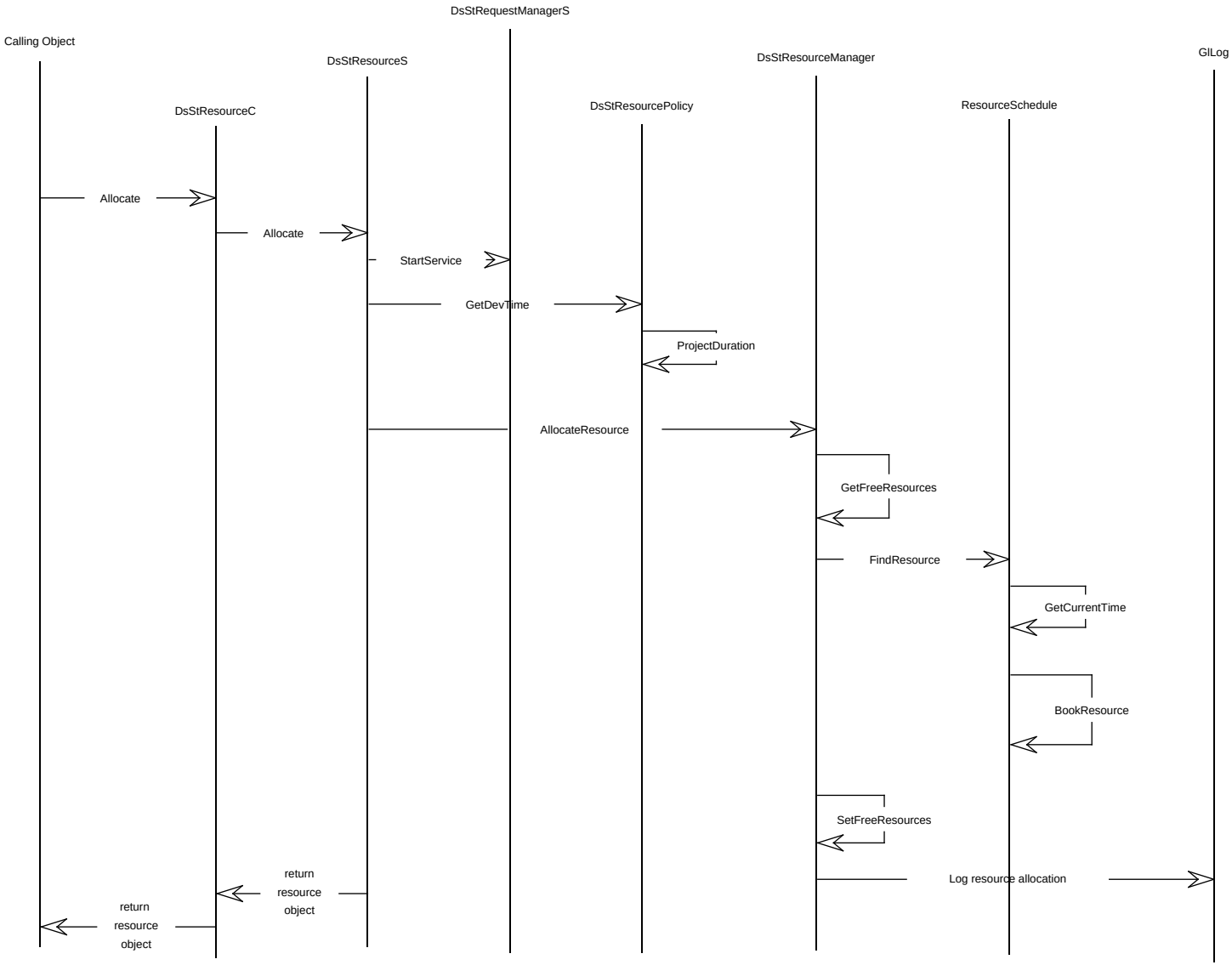


Figure 6.4-4. ALLOCATEPHYSRES_RESAVAIL Dynamic Model

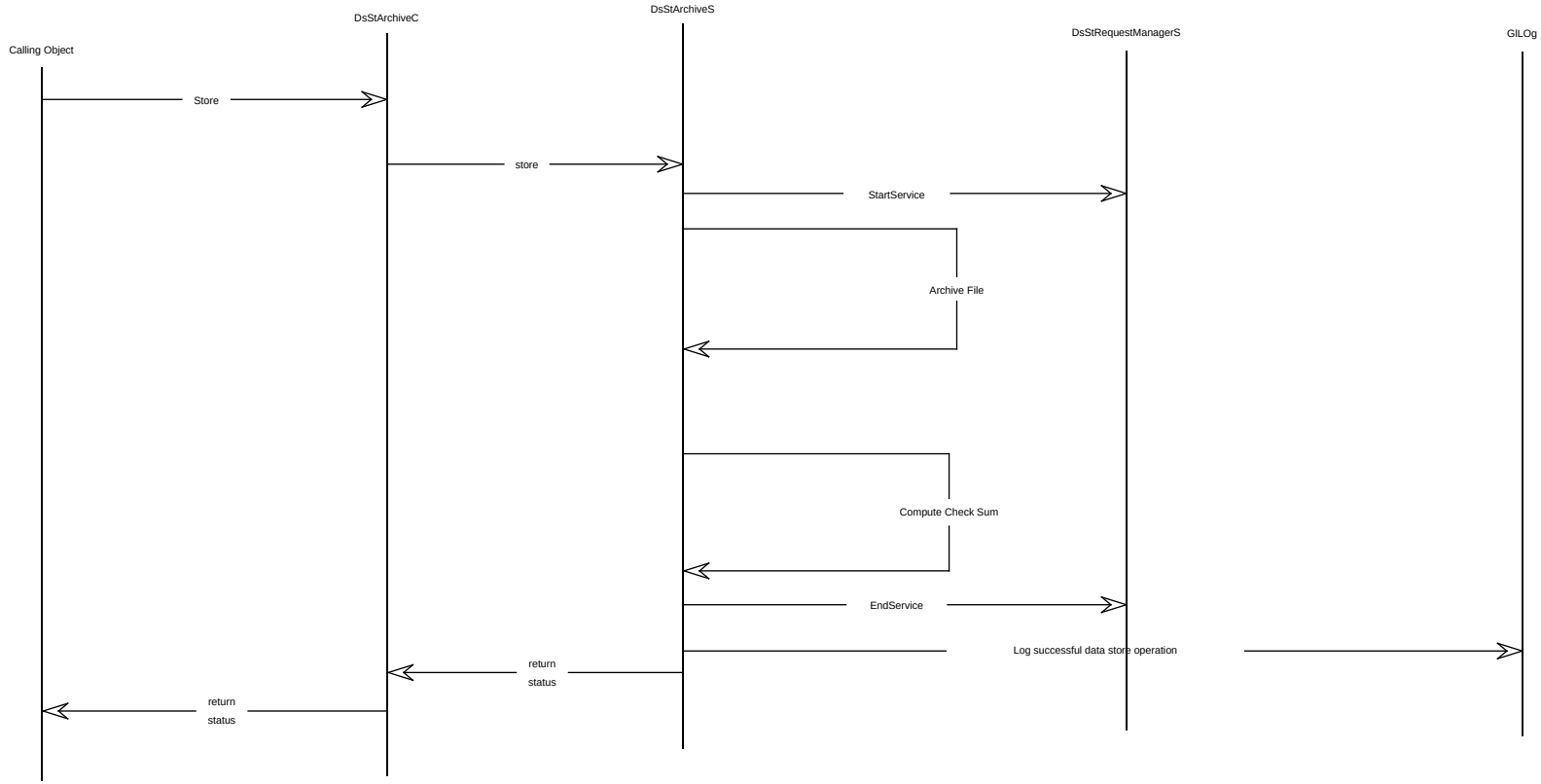


Figure 6.4-5. ARCHIVE_INSERT Dynamic Model

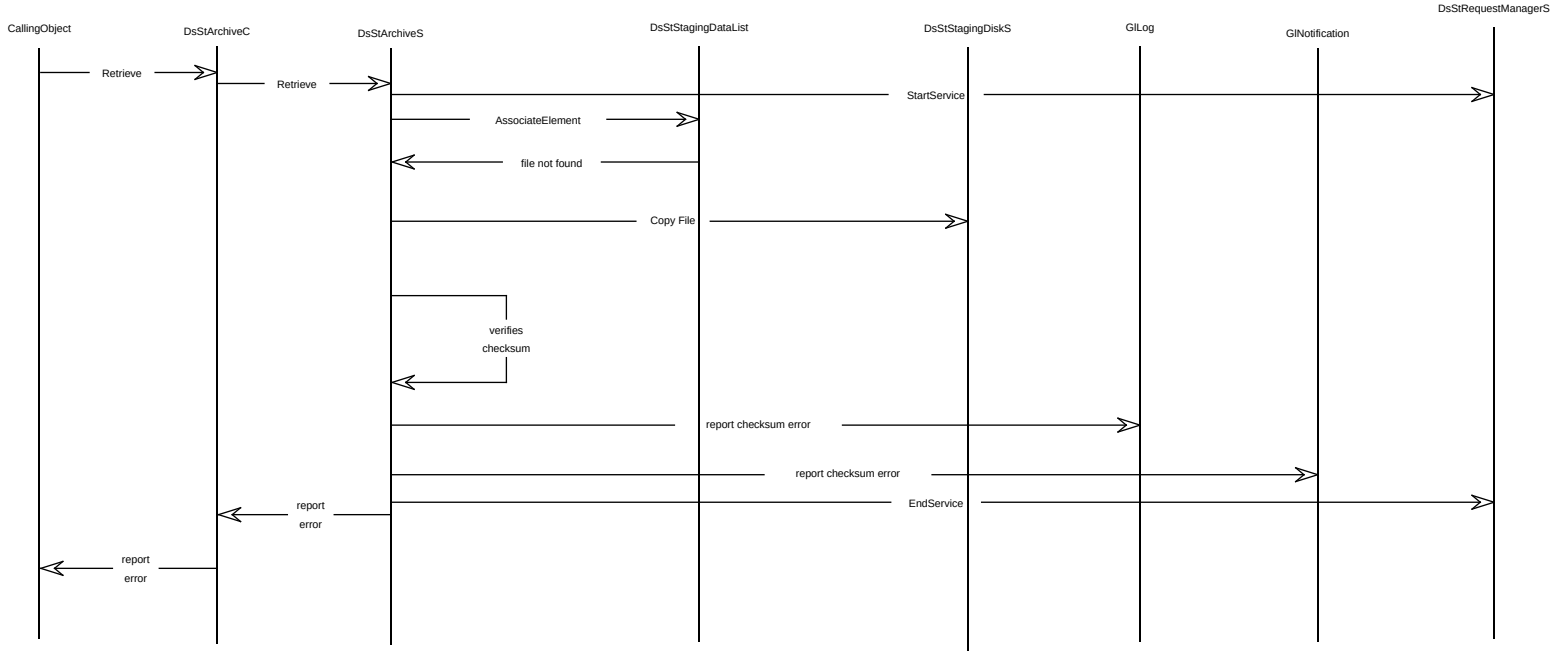


Figure 6.4-6. ARCHIVE_RETRIEVE_CHECKSUMERROR Dynamic Model

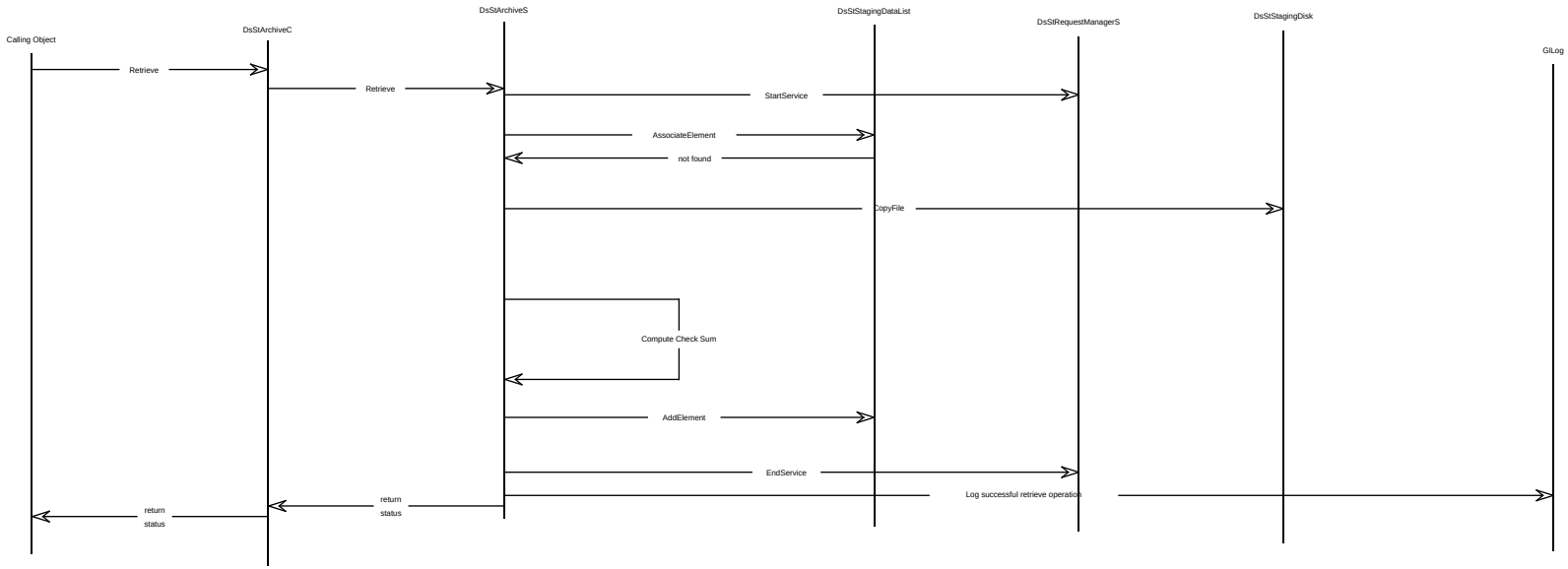


Figure 6.4-7. ARCHIVE_RETRIEVE_NOCACHEHIT Dynamic Model

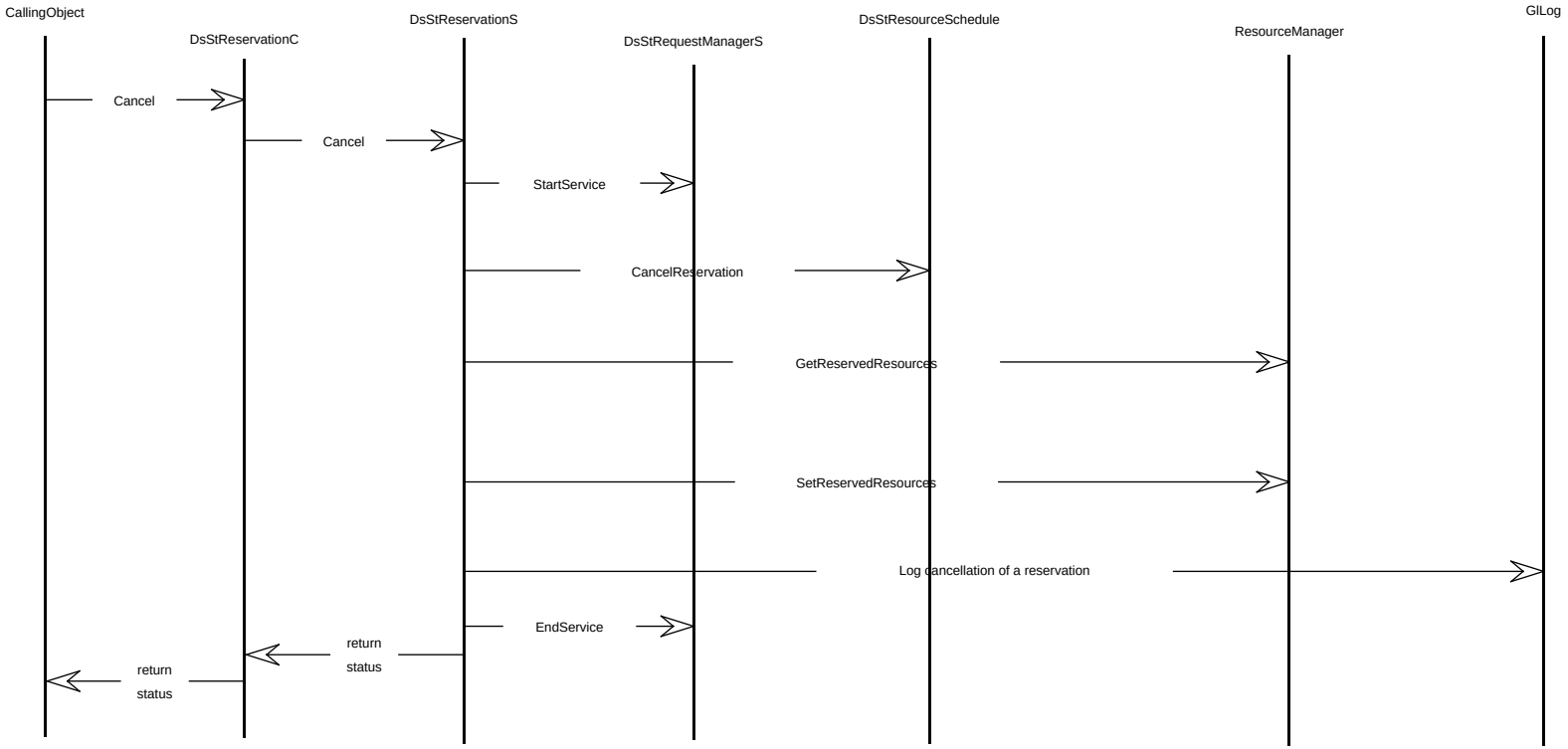


Figure 6.4-8. CANCEL_RESERVATION Dynamic Model

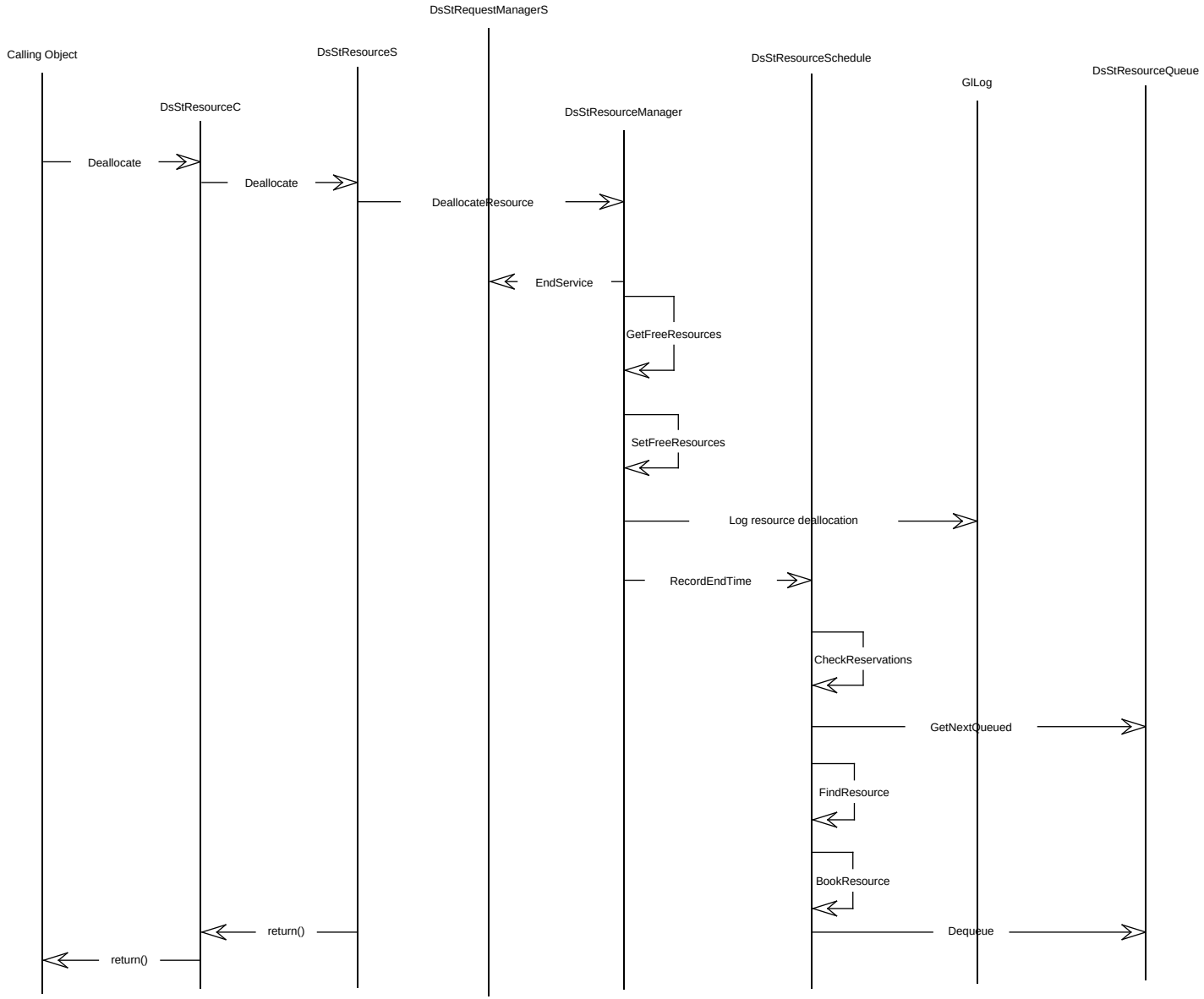


Figure 6.4-9. DEALLOCATEPHYSRES_QUEUED_REQUESTS Dynamic Model

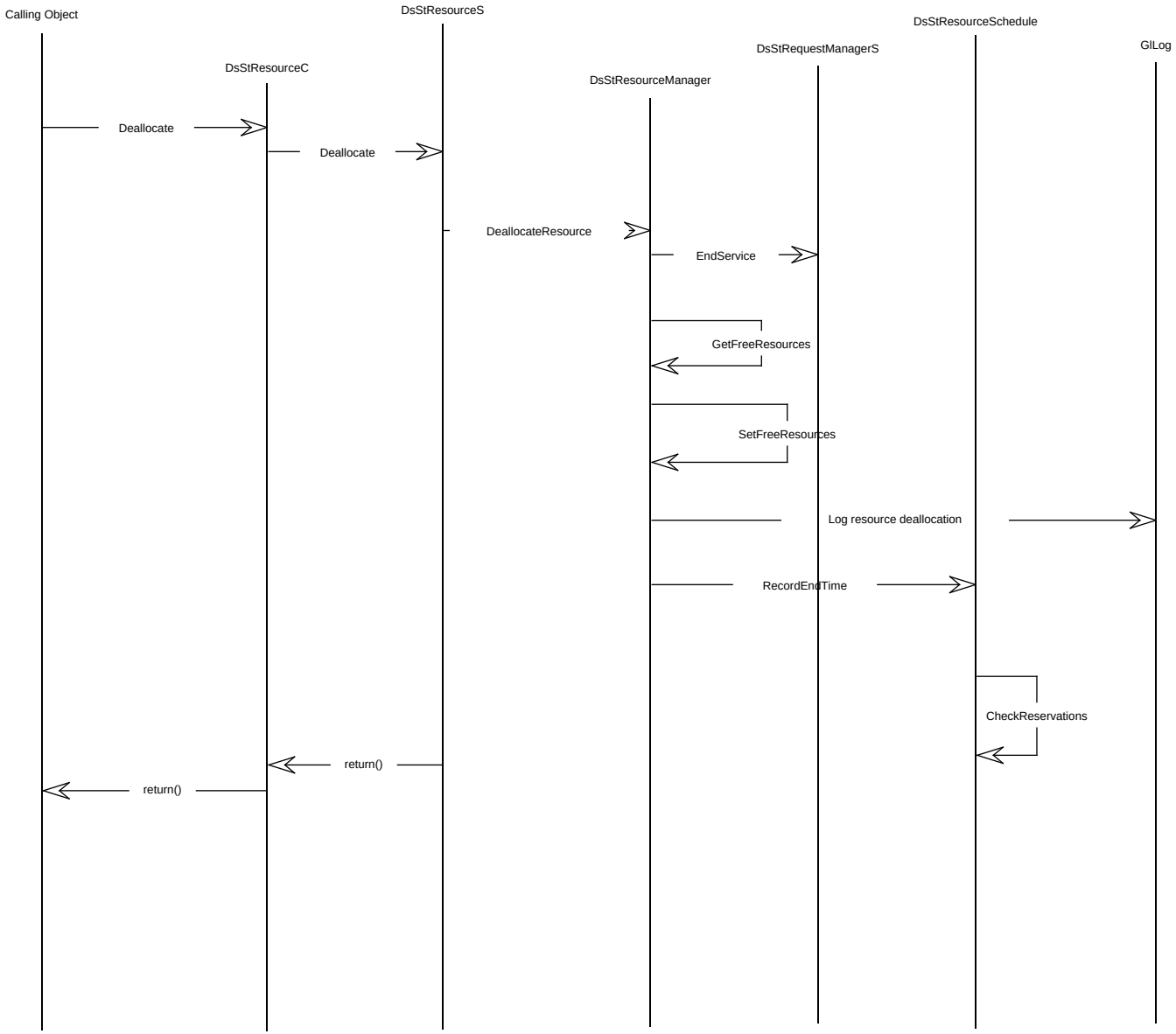


Figure 6.4-10. DEALLOCATEPHYSRES_RESERVATION Dynamic Model

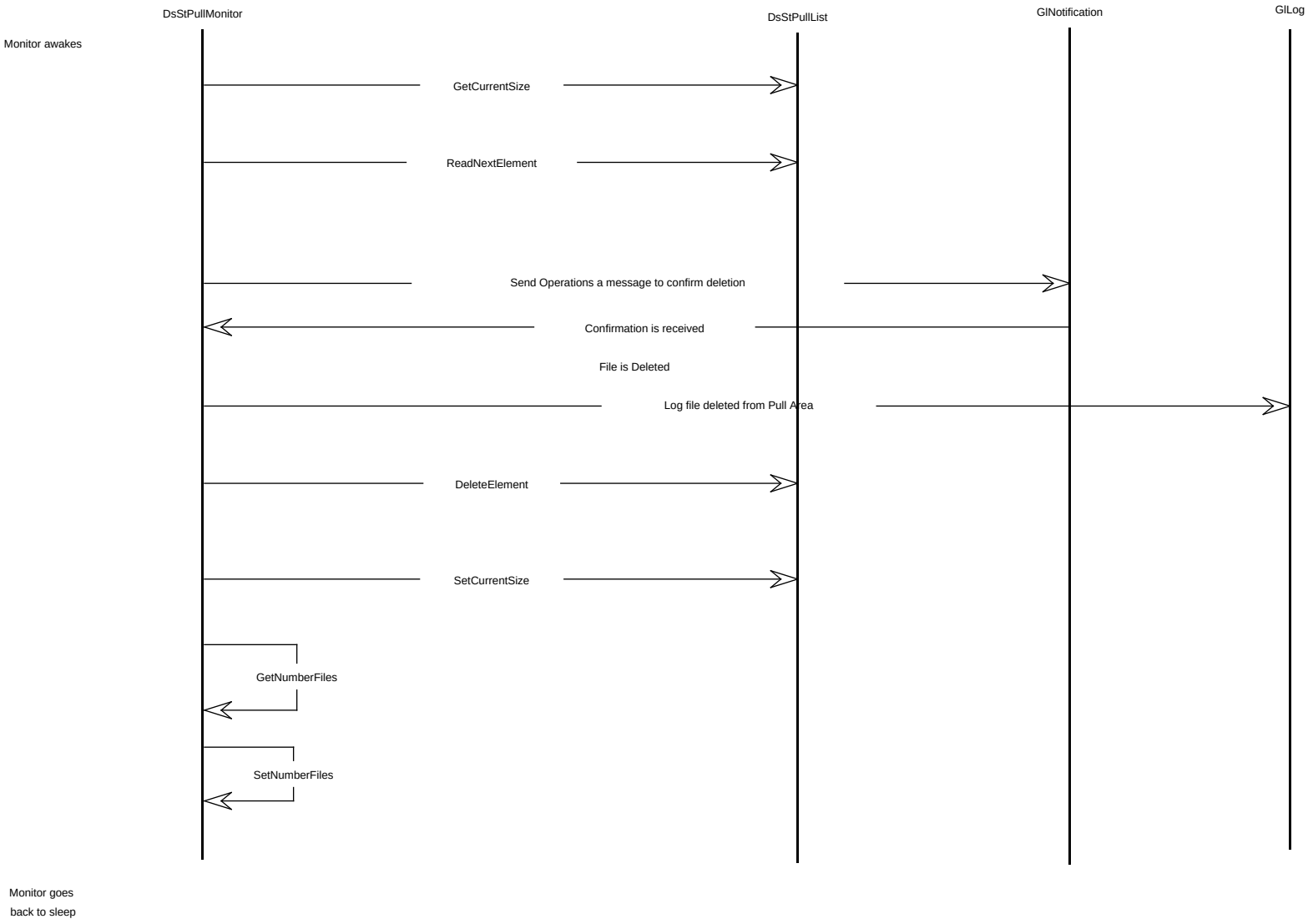


Figure 6.4-11. DELETE_PULLDISKDATA_CONFIRMED Dynamic Model

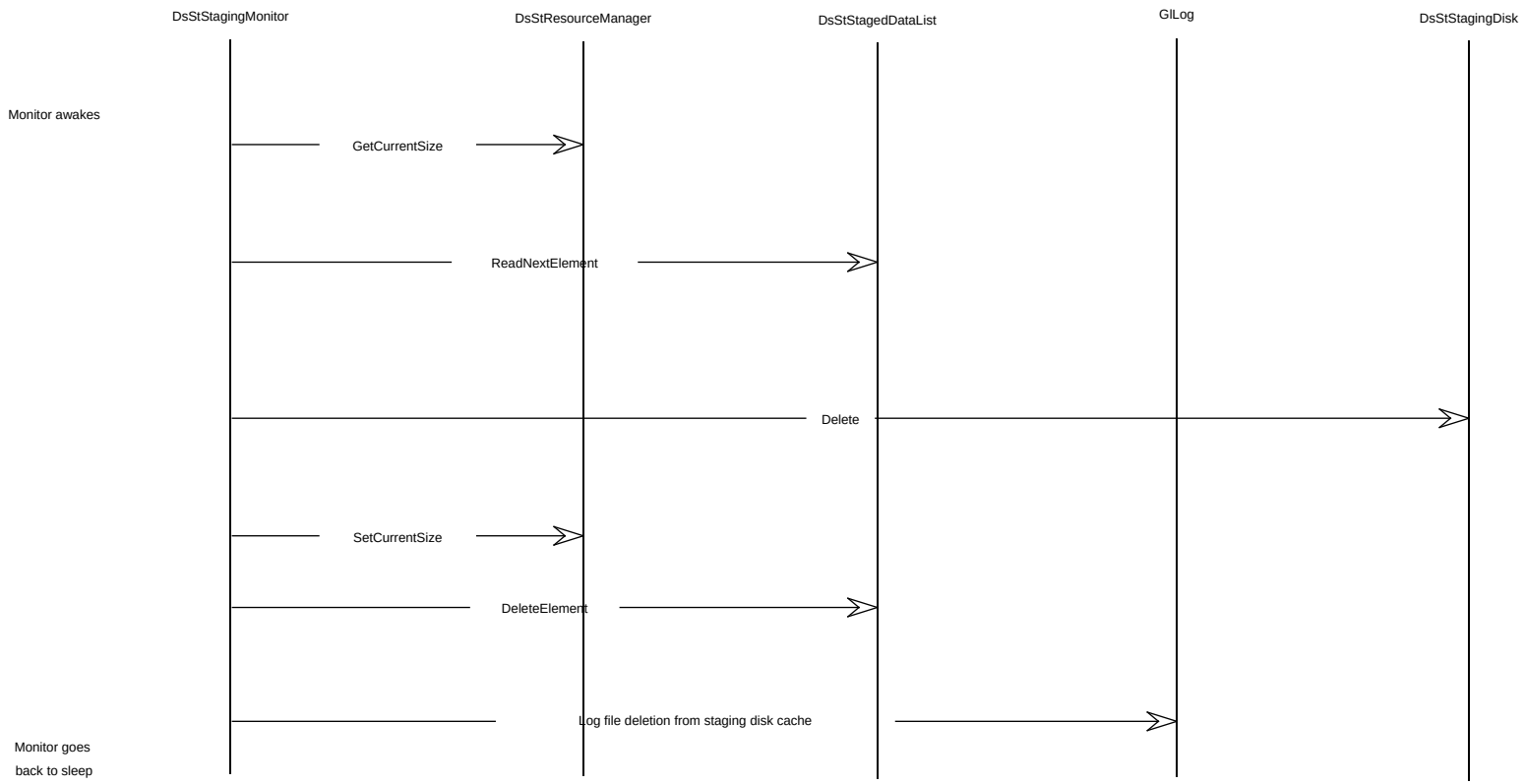


Figure 6.4-12. DELETE_STAGINGDISKDATA Dynamic Model

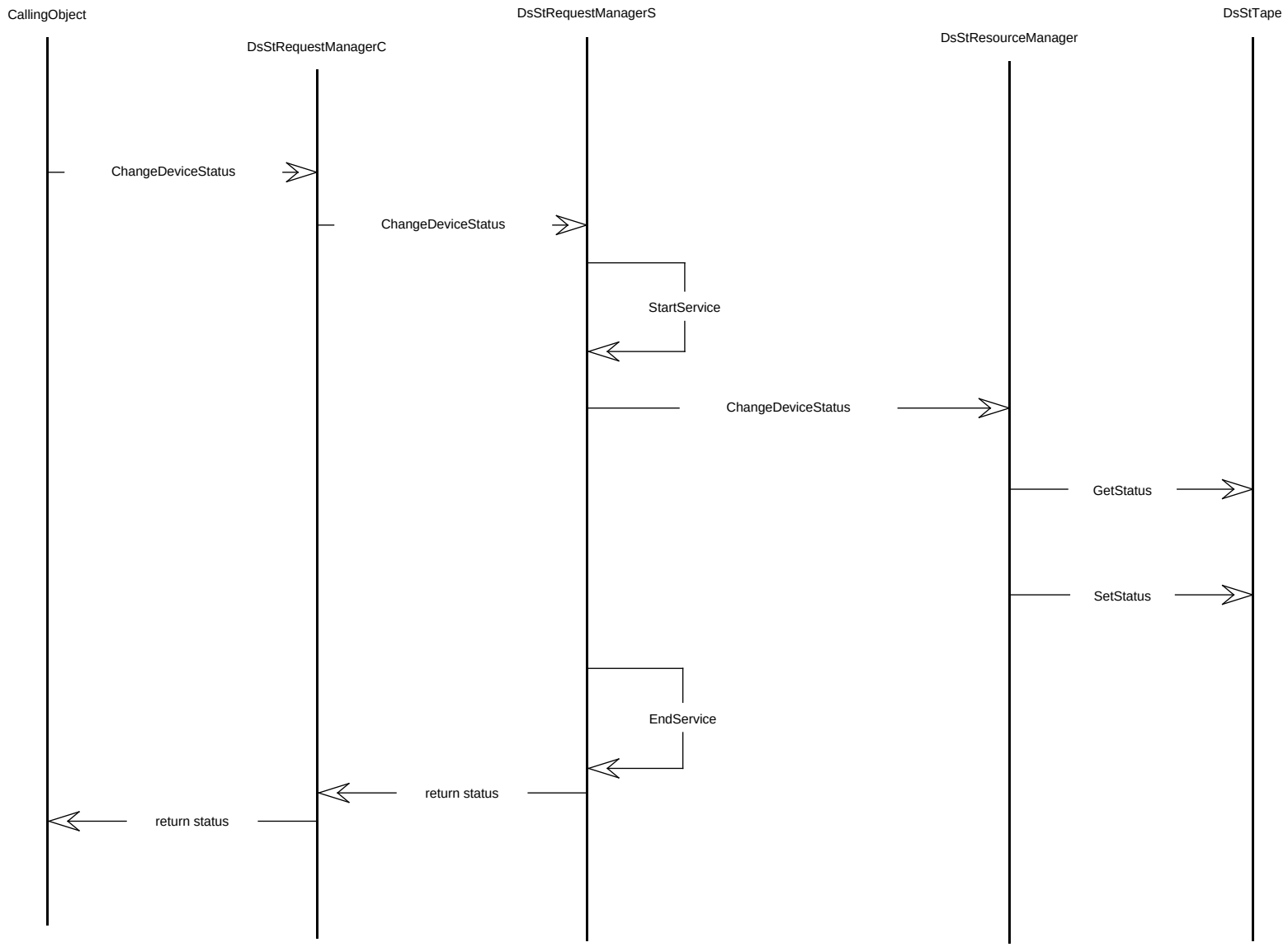


Figure 6.4-13. SET_TAPESTATE Dynamic Model

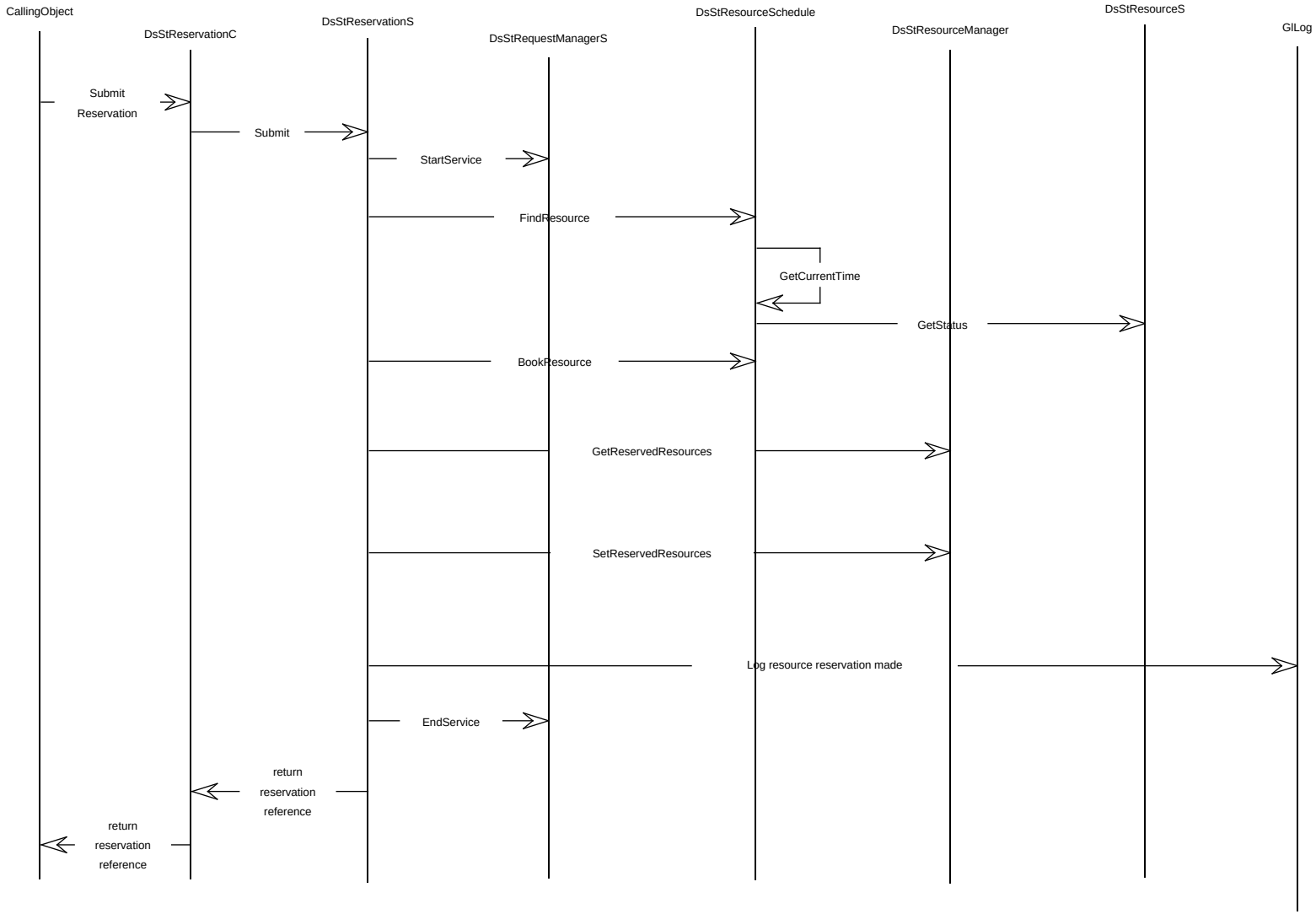


Figure 6.4-14. SUBMIT_RESERVATION Dynamic Model

6.5 CSCI Structure

Table 6.5-1 shows the components (CSCs) of the CSCI. Each CSC is described and designated as being custom developed code (DEV), off-the-shelf (OTS) or a combination of the two (DEV/ OTS). If the custom developed code will be used for integration purposes, it is identified as WRAPPER.

Table 6.5-1. STMGT's Components

CSC	Description	Type
Service Clients	Provides client interface to ECS archives and the peripheral devices used for data ingest and distribution	DEV/OTS
Resource Management	Provides capability to manage all resources of the Data Server Subsystem	DEV
Data Storage	Provides capability to store and provide access to earth science data	DEV/OTS
Peripherals	Provides access to devices used for data ingest and distribution	DEV
File	Provides file level access to data on staging disk	DEV

6.5.1 CSC Definitions

6.5.1.1 Service Clients CSC

Purpose and Description

All of the classes contained in this CSC, with the exception of one (DsStArchive) are products custom developed by the Storage Management CSCI. These classes collectively provide an interface to Data Server Subsystem resources and are categorized into four different groups, each providing an interface for a different type of client.

Classes

Archive Clients will primarily want access to the ECS archives for storing or retrieving data files. The SDSRV and DDSRV CSCIs are placed in this category.

The following are classes contained in the Archive Client CSC:

DsStArchive
DsStStagingFile
DsStStagingDisk
DsStResource
DsStStorageResource

The DsStArchive class encapsulates the AMASS File Storage Management System OTS product.

Peripheral Clients will similarly want access to the peripheral devices managed by the Storage Management CI. The INGST Subsystem and the DDIST CSCI are placed in this category. The following classes are contained in the Peripheral Clients CSC:

DsStResource
DsStPhysicalResource
DsStTape
DsStCDROM

DsStPrinter
DsStFax
DsStNetworkResource
DsStReservation

The DsStNetworkResource class encapsulates the CSS supplied API which supports the OTS FTP product.

In addition to their primary needs for either access to data files in the ECS archive or devices which will be used to ingest or distribute data files, clients will also have a need for storing data, hence, the third category of client, Storage Resources. The following classes are contained in the Storage Resources Client CSC:

DsStResource
DsStStorageResource
DsStStagingDisk
DsStArchive
DsStFileDirectory

The fourth client category, OP-ADMIN, reflects the needs of Operations staff and/or administrative users to interface with the Storage Management CSCI. The following classes are contained in the OP-ADMIN Client CSC:

DsStResourceManager
DsStResourceQueue
DsStResourceSchedule
DsStResourcePolicy
DsStRequestManager

Candidate products

Not Applicable

ECS white paper references

Not Applicable

6.5.1.2 Resource Management CSC

Purpose and Description

The following classes provide the capability to manage all resources of the Data Server. The classes contained in this CSC are products custom developed by the Storage Management CSCI.

Classes

DsStResourceManager
DsStResourceQueue
DsStResourceSchedule
DsStResourcePolicy
DsStStagingDataList
DsStStagingMonitor
DsStPullList
DsStPullMonitor

DsStResource
DsStReservation
DsStSCacheConfig
DsStPullConfig
DsStSchedulingConfig
DsStResourceConfig

Candidate products

Not Applicable

ECS white paper references

Not Applicable

6.5.1.3 Data Storage CSC

Purpose and Description

The classes in this CSC provide the capability to store data on a persistent or non-persistent basis and access it. These classes, with the exception of DsStArchive as indicated previously, are products custom developed by the Storage Management CSCI.

Classes

DsStResource
DsStStorageResource
DsStStagingDisk
DsStArchive
DsStStagingFile

Candidate products

Not Applicable

ECS white paper references

Not Applicable

6.5.1.4 Peripherals CSC

Purpose and Description

The classes in this CSC represent the devices used to ingest and distribute data. These classes, with one exception (DsStNetworkResource), are products custom developed by the Storage Management CSCI. The DsStNetworkResource class encapsulates the CSS supplied API which supports the OTS FTP product.

Classes

DsStResource
DsStPhysicalResource
DsStPrinter
DsStTape
DsStFax
DsStCDROM
DsStNetworkResource

Candidate products

Not Applicable

ECS white paper references

Not Applicable

6.5.1.5 File CSC

Purpose and Description

The classes in this CSC provide clients file granularity level of access to data files on staging disk. These classes are products custom developed by the Storage Management CSCI.

Classes

DsStDirectory
DsStFile
DsStStagingFile

Candidate products

Not Applicable

ECS white paper references

Not Applicable

6.5.2 CSCI Dynamic Architecture

Storage Management will consist of two distinct types of processes: resource managers, and disk monitors. There are two disk monitors, a staging monitor, and a pull monitor.

ResourceManager Processes

The Resource Manager processes will be responsible for maintaining a schedule of current and proposed resource activity and for distributing resource access keys which will allow resource access for a specific period of time. Each type of resource pool will have its own resource manager. Currently, five types of resource manager processes have been identified: network, staging disk, tape, CD ROM, and printers. The ResourceManager process comprises the following CSCs:

Resource Management
Service Client
OP-Admin
Peripherals

StagingMonitor Process

The Staging Monitor will be responsible for managing the group of data files that have been retrieved from the archive and placed into a cache area on staging disk. A list of these data files will be maintained so that subsequent data retrieval requests can be fulfilled immediately without requiring an additional archive access. The Staging Monitor will also be responsible for deleting old or seldom accessed files in order to prevent the cache area from becoming too full. Several potential (quite sophisticated) algorithms are available for monitoring and maintaining the data levels at a pre-set capacity. The StagingMonitor process comprises the following CSC s:

File
Data Storage
Resource Management

PullMonitor Process

The Pull Monitor will be responsible for managing the files in the user pull area. As files are retrieved (i.e., electronically pulled) from the user pull area or as the files become stale (their time-out periods have expired) then the Pull Monitor will delete them. The PullMonitor process comprises the following CSCs:

Service Clients
Peripherals
Resource Management
File

6.6 CSCI Management and Operation

The materials in the following paragraphs discuss the management and operations of software components discussed in section 4.5.

6.6.1 System Management Strategy

The STMGT CSCI is designed to provide robust data storage, retrieval, and distribution services to external data providers and requesters. Specifically, the design goal of the STMGT CSCI is to always return status (successful or unsuccessful) for every received request. To accomplish that goal, the CSCI follows ECS project guidelines for:

- Process startup and shutdown;
- Error detection and reporting;
- Fault tolerance and error recovery

6.6.1.1 Startup/Shutdown

MSS provides life-cycle services for system startup and shutdown. The StorageResourceManagement SDSRV processes act as an “object factory”. As such each process instantiates objects in process threads (pthreads) when a request is serviced. At STMGT system startup, seven required processes are launched: 5 StorageResourceManagement processes (one per *type* of resource), StagingMonitor, and PullMonitor. The StorageResourceManagement processes are started as standalone processes. The primary role of the StorageResourceManagement processes are to maintain a schedule of current and proposed resource activity and to distribute resource access keys which will allow resource access for a specific period of time. Each type of resource pool will have its own StorageResourceManager. Currently, five types of StorageResourceManager processes have been identified: network, staging disk, tape, CD ROM, and printers.

The StagingMonitor process will also routinely be started at system startup. This process will be responsible for managing the files that have been retrieved from the archive and placed into a cache area on staging disk. A list of the data files in this cache will be maintained so that subsequent requests can be fulfilled immediately without requiring an additional archive access. The

StagingMonitor will also be responsible for deleting old or seldom accessed files in order to prevent the cache area from becoming too full. Several potential (quite sophisticated) algorithms are available for monitoring and maintaining the data levels at a pre-set capacity.

The Pull Monitor will be responsible for managing the files in the user pull area. As files are retrieved from the user pull area or as the files become stale (their time-out periods have expired) the Pull Monitor will delete them.

6.6.1.2 Error Detection and Reporting

The STMGT CSCI is designed for primarily automated operations with little need for operations involvement short of tuning and critical error conditions. CSS and MSS jointly provide event logging services for logging and reporting errors and faults, for browsing error/status logs, and for detecting and reporting critical errors. The STMGT CSCI will fully use those services during operations. Errors/status may be reported in two error logs. MSS maintains the first log, the MSS event log. It contains errors/status of interest to operations staff to evaluate system status and to perform trend analysis. The STMGT subsystem maintains the second log. The STMGT event log contains selected errors/status from the MSS event log (for context) plus highly-detailed debug events. Software maintenance personnel use the STMGT event log to diagnose system and software problems in response to trouble tickets.

Non-critical errors encountered during processing that will be handled at the application level will be fully resolved and enumerated during development. Major conditions that require operator intervention and/or are considered catastrophic in the processing of requests are listed in Table 6.6.1.2-1.

6.6.1.3 Fault Tolerance and Error Recovery

Once a service request is accepted from a client process (client being defined as any service requester), it is the STMGT design goal to complete the request processing and return status (successful or unsuccessful) to the requester. The STMGT CSCI is built on the model of check pointing processing at four points: the request, the staging data list, pull data list, and resource tables. During restart or recovery operations the CSCI will restore all checkpointed data and resume processing at the last unprocessed command. In the case of data retrieval, any data that has been staged will be identified via the staging data list and not be retrieved a second time. Resources that have been allocated to other processes will remain in that state until deallocation. Pull data will be checked at startup for expiration and clean up operations.

Failure scenarios with recovery methods:

a. Failure of a STMGT executable. Any failed process is immediately restarted as a Unix standalone process. The persistent data associated with the process is restored from their checkpointed states. Any requests outstanding are then executed.

b. Loss of the data base tables used for check pointing. The data base used for persistent data is actually outside of this CSCI. The data base management system will automatically log transactions to allow restorations of table information. This feature, coupled with DBA generated Data Base backups, provide for recovery. Since high reliability is required in this area, the Data Base tables will be stored on RAID.

Table 6.6.1.2-1. STMGT Error Categories

Error Category	Actions to Be Taken
Initialization File/ Environment Corrupt	This would be seen during a system startup process and would result in one or more executables not starting. Operations staff evaluate the condition and correct.
Unable to archive data	Internal Data Server fault. Log errors to the event log and return status to the external data provider. Report alert to operations staff. Operations staff evaluate errors off-line and request re-ingest as necessary. (Note: Data Server will re-vector ingested data to a different device in the event of a single device failure.)
Unable to read peripheral media	Internal Data Server fault. Log errors to the event log and return status to the external data provider. Report alert to operations staff. Operations staff evaluate errors off-line and request re-ingest as necessary. (Note: if a peripheral device fails, Data Server will revector the media to a different peripheral.)
Unable to transfer data to be archived	After a system-tunable number of retries, log errors to the event log and return status to the external data provider. Report alert to operations staff. Operations staff evaluate errors off-line to evaluate and correct communications network problems.
Internal queue overflow	Errors reported along these lines represent a very poorly tuned and/or faulty system. This type error would represent potential loss of service requests. Operations staff would immediately throttle back system processing thresholds and write a trouble ticket for future off line analysis and tuning.
Unable to allocate disk space	Unable to allocate working storage space using Data Server STMGT CSCI services. This is another symptom of a poorly tuned system. Report alert to operations staff who would immediately lower system thresholds for requests. Operations staff would analyze system off-line and tune.
Unable to set up external data requester session	Limit exceeded for allowable number of external data provider sessions. Log errors to the event log and return status to the external data provider, indicating that the session connection should be re-attempted later. (Note: based on the modeled transaction load, this error condition is expected to occur very rarely, if at all.)

c. Failure of the processor on which a STMGT process is running. In general, the processor automatically restarts. Restart of individual processes is handled as a combination of one or more of the above process restarts. If the processor is disabled, the disablement is detected by MSS SNMP services and a backup processor is restarted. The backup processor has full access to the data base tables used for check pointing. Again, restart of individual processes is handled as described above.

d. Failure of an external application. After a given number of retries to transmit data or via DCE services that alert us of failure of the recipient process, operations staff are notified by means of an alert message. The DAAC operations staff will coordinate to diagnose the failure.

6.6.2 Operator Interfaces

DAAC operations personnel are provided with an X-Windows/Motif-based GUI to access operations data bases and system configuration information. Table 6.6.2-1 highlights the critical STMGT GUI screens.

Table 6.6.2-1. Storage Management GUI

GUI	Description	Data	Operations
Storage Management GUI	Primary Screen for Storage Management GUIs	References to: Request, Log & Reports, Configuration, Client Management and Resource Management GUIs.	Logon Realizes and/or makes the GUIs visible.
Archive Administration GUI	Allows operators to manage non-COTS storage resources	Resource Utilization information	View Backup Restore Retrieve
Device Control GUI	Allows operators to manage various available media devices	Device information Device alarms	View Mount Dismount Bring on-line Put off-line
COTS Archive GUIs	COTS provided GUI(s) that support management of the COTS archive product	TBD	TBD

6.6.3 Standard STMGT Reports

Table 6.6.3-1. Standard Storage Management Reports (1 of 2)

Report Type	Report Description	Intended Audience
Received Service Requests	Provides a chronological listing of requests. Each entry includes: Request Identifier, date/time stamp, requested operation, and request completion status.	System Operator Performance Analyst Operations Supervisor DAAC Manager
Device Failure	Provides a Chronological listing of device failures. Each entry includes: device identifier, date/time stamp, error code, failure message.	System Operator Maintenance Technician Maintenance Supervisor Operations Supervisor DAAC Manager
Archive Activity Log	Provides a chronological listing of all archive activities, requests, and failures.	System Operator Performance Analyst Operations Supervisor DAAC Manager
Selected Archive Activity	Based on the information in the Archive Activity Log, this report allows operations personnel to select, and sort log entries by start/stop time, operation requested, and/or request completion status.	System Operator Performance Analyst Maintenance Technician Maintenance Supervisor Operations Supervisor DAAC Manager

Table 6.6.3-1. Standard Storage Management Reports (2 of 2)

Report Type	Report Description	Intended Audience
Intermediate Activity Log	Provides a listing of the resource activity that occurs between the receipt of a Request and Request Completion. Information includes: a date/time stamp, operation identifier, affiliated request identifier, disk space utilization (allocate/ deallocate), media utilization (mount/ dismount), file utilization, (read/write/copy/ delete)	System Operator Performance Analyst Operations Supervisor DAAC Manager
Selected Intermediate Activity	Based on the information in the Intermediate Activity Log, this report allows operations personnel to select, and sort log entries by start/stop time, operation id, affiliated request identifier,	System Operator Performance Analyst Operations Supervisor DAAC Manager
Inventory Update Report	Provides a record of new data files added to the DAAC, and provides summary information associated with the addition of new data files based on the reporting period which is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date. Associated information includes: number of requests received, number of successful and unsuccessful requests, volume of data archived, average volume of data archived per request, current number of archived files, and average number of files per request.	System Operator Performance Analyst Operations Supervisor DAAC Manager
Archive Resource	Provides a current status of utilized and available resources within the archive. Report includes: a date/time stamp, the total unallocated media volumes by type and capacity, and the total number of unpopulated media locations (slots) for each archive unit.	System Operator Performance Analyst Operations Supervisor DAAC Manager
Archive Error	Is a summary report of the frequency of errors encountered during archive processing. The reporting period is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date. Information for each unsuccessful request includes: date/time stamp, request identifier, error code, error message.	System Operator Performance Analyst Operations Supervisor DAAC Manager
Pull Area Utilization	Provides information on the total staged data volume in Gigabytes, and the number of product instances staged.	System Operator Sr. Science Coordinator Resource Planner Performance Analyst Operations Supervisor DAAC Manager

This page intentionally left blank.

7. DDIST - Data Distribution CSCI

7.1 CSCI Overview

The Data Distribution CSCI provides the Data Server and Operations users the capability to monitor and control processing for distribution requests. Data Distribution processing mainly consists of preparing requested data objects for distribution on specified media or via the network and subsequently delivering or causing the delivery of data products to requesting clients.

In addition to preparing the data, Data Distribution will also generate the necessary packaging materials if the data is to be distributed on media and distribution metadata if the data is to be distributed via the network. The packaging materials include the packing list, showing all data objects stored on the delivery media. The media will include CD ROM and tape of various formats (8mm, and 6250 bpi 9-track).

7.2 CSCI Context

Figure 7.2-1 provides an event flow diagram for this CSCI, which shows the interfaces with other CSCIs; Table 7.2-1 summarizes the flow depicted in Figure 7.2-1.

Table 7.2-2 provides a mapping for the DDIST CI interfaces provided to other ECS CSCI's. The table defines the interfaces in terms of the DDIST-provided classes (from the Client CSC), and the specific service to be used to accomplish that interface. The response (return parameter) from that member function is also included.

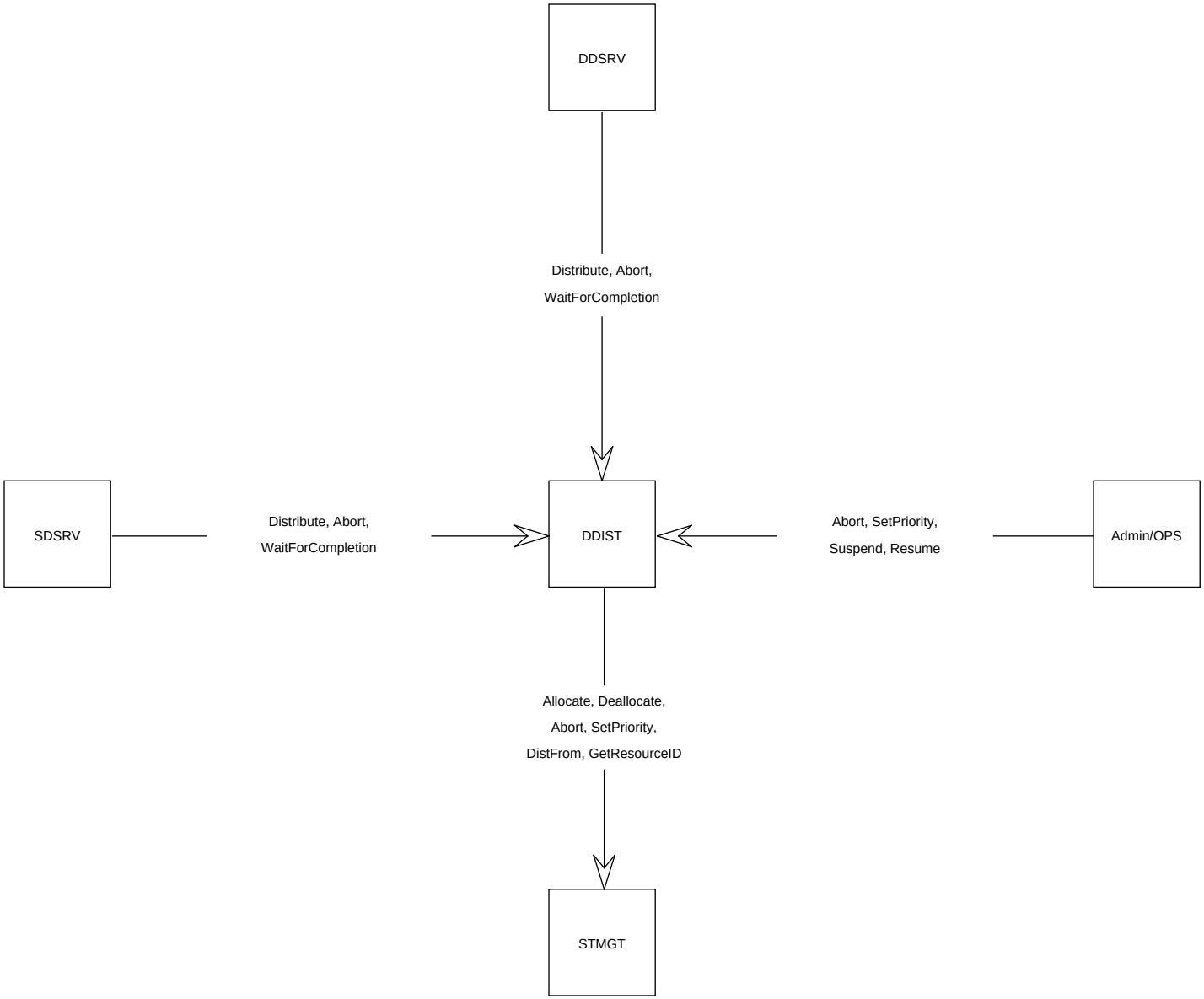


Figure 7.2-1. DDIST Event Flow Diagram

Table 7.2-1. DDIST Event Flow Summary Table

Sender	Receiver	Event Name
DDIST	STMGT	Abort
SDSRV	DDIST	Abort
DDSRV	DDIST	Abort
Admin/OPS	DDIST	Abort
DDIST	STMGT	Allocate
DDIST	STMGT	Deallocate
DDIST	STMGT	DistFrom
SDSRV	DDIST	Distribute
DDSRV	DDIST	Distribute
DDIST	STMGT	GetResourceID
Admin/OPS	DDIST	Resume
DDIST	STMGT	SetPriority
Admin/OPS	DDIST	SetPriority
Admin/OPS	DDIST	Suspend
DDSRV	DDIST	WaitForCompletion
SDSRV	DDIST	WaitForCompletion

Table 7.2-2. DDIST CI Interfaces

Interface	Input Data	Output Data	Description
Distribute	Media Type, User Profile	Distribution Request	Request distribution of the items specified in the distribution list. SDSRV will create the list of items to be distributed and invoke this service to start the distribution; the service asynchronously returns a distribution request, which can then be used to perform additional operation upon the request.
WaitFor Completion	none	Completion Status	Wait for a distribution request to complete. This service will block the calling thread.
Abort	none	Completion Status	Abort processing of a distribution request. If WaitForCompletion has also been called, this service would be called from a separate thread.
SetPriority	Priority	none	Change the priority of a distribution request.
Suspend	none	Status	Suspend processing of a distribution request.
Resume	none	Status	Resume processing of a suspended distribution request.

7.3 CSCI Object Model

This section provides object models for the Data Distribution CSCI. Two models are presented, DsDdInterfaceClasses and DsDdManager. Some classes appear on the diagrams for both models.

Figure 7.3-1, DsDdInterfaceClasses, depicts classes which are used by, and visible to, other CSCIs. Interface classes may be, but are not necessarily, implemented as distributed objects. Classes which become distributed objects, identified on the diagrams via the annotation [DISTR OBJ], have both a client and a server representation which acts as a single object, i.e. an operation requested via the client object results in action upon the server object. Distributed objects have client and server specializations (designated via a suffix of C or S, respectively, for the class name) of a common base class. Additionally, a distributed object may have different presentations for different clients; for example, because an operator can perform operations which are not available to a user, separate operator and user client specializations exist (DsDdOpsRequestC and DsDdDistRequestC, respectively).

While the server specializations are not directly visible to other CSCIs, they are provided on the DsDdInterfaceClasses diagram to promote understanding of the interfaces.

Figure 7.3-2, DsDdManager, depicts the classes involved in the distribution server; as such, it contains (in addition to its other classes) server specializations of distributed objects, which are also shown on DsDdInterfaceClasses.

General Notes: In the following object class descriptions, classes always inherit attributes, operations, and associations from their parent classes. Where all attributes in the current design are inherited from the parent class, this is indicated by the text "*All Attributes inherited from the parent class*" (and analogously for operations). If a derived class has additional attributes of its own, those new attributes are listed, but the attributes from the parent class are not repeated (analogously for operations).

The Object Classes in the Object Model diagrams and descriptions each have an identified set of attributes and public operations. In most cases the listed public services do not include two operations: a constructor (often referred to as "Create") and a destructor (often referred to as "Destroy"). For most Object Classes in these models the constructor and destructor operations are not listed; however, they are shown if they perform special processing beyond normal construction and destruction, or if they are overloaded to allow for differing signatures.

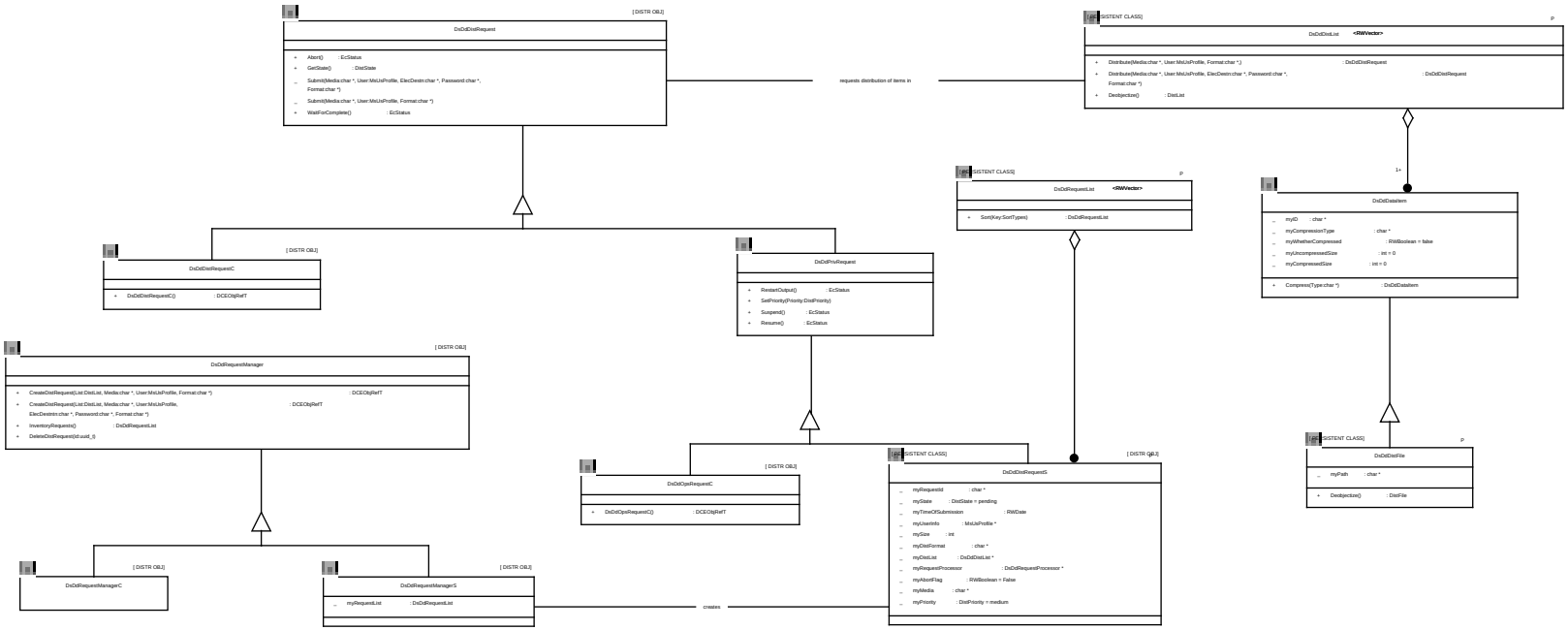


Figure 7.3-1. DsDdInterfaceClasses Object Model Diagram

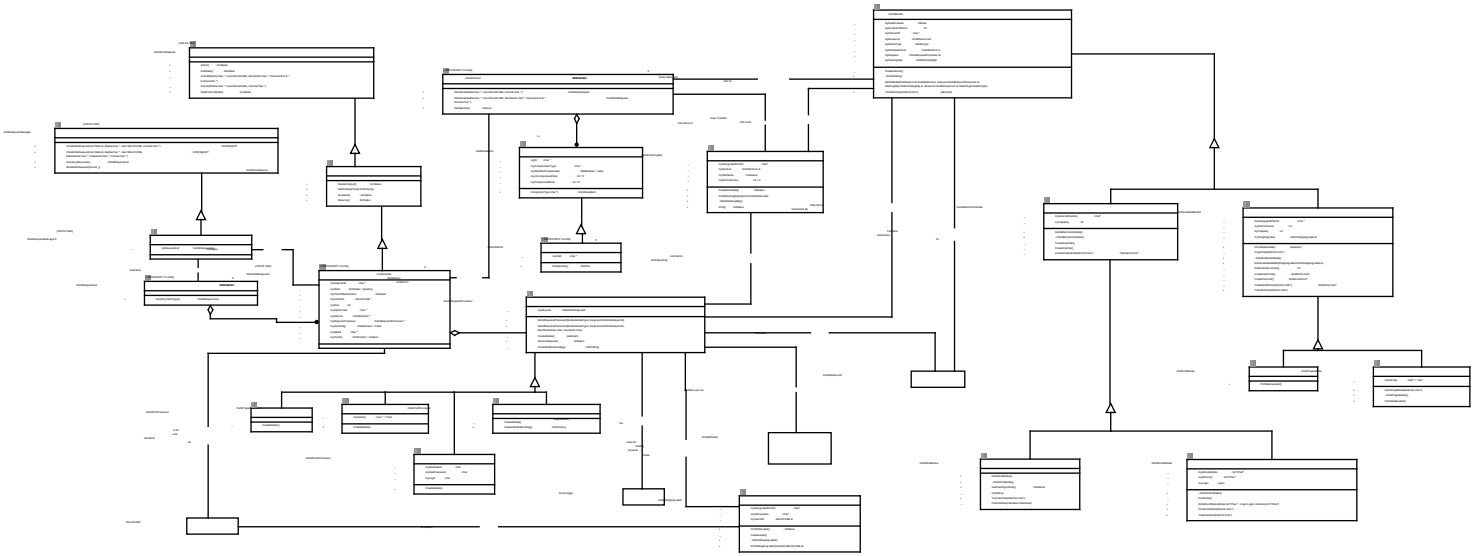


Figure 7.3-2. DsDdManager Object Model Diagram

7.3.1 DsDdCDMedia Class

Parent Class: DsDdLabeledMedia
Public: No Distributed Object: No
Purpose and Description:
Initiates and monitors the transfer of data items via cpio or tar archive file to a CD resource.

Attributes:

All Attributes inherited from parent class

Operations:

PrintMediaLabel - This operation will be implemented in release B. It will be a polymorphic operation which depends on the type of device for which the label is being printed.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdCDMedia class has associations with the following classes:
None

7.3.2 DsDdCDProcessor Class

Parent Class: DsDdRequestProcessor
Public: NoDistributed Object: No
Purpose and Description:
Defines the concrete CreateMedia to perform the correct processing of distribution via CD.

Attributes:

All Attributes inherited from parent class

Operations:

CreateMedia - Creates the instances of the objects needed for interfacing with the devices needed for transfer of the data items
Arguments:
Return Type: Void
Privilege: Private

Associations:

The DsDdCDProcessor class has associations with the following classes:
None

7.3.3 DsDdDataItem Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class:
Purpose and Description:
Base class for items to be distributed. For now only a file specialization of this class exists and is used, but the base class is defined to support possible future use of streams et al.

Attributes:

myCompressedSize - Size of the object after being compressed.
Data Type: int
Privilege: Private
Default Value: 0

myCompressionType - Type of compression, if any, performed on the item. Compression types are TBD, and no compression is supported in Release A.
Data Type: char *
Privilege: Private
Default Value:

myID - Identifier, such as filename, for the data item.
Data Type: char *
Privilege: Private
Default Value:

myUncompressedSize - Size of the object before it was compressed.
Data Type: int

Privilege: Private
Default Value: 0

myWhetherCompressed - Whether the object is compressed.
Data Type: RWBoolean
Privilege: Private
Default Value: false

Operations:

Compress
Arguments: Type:char *
Return Type: DsDdDataItem
Privilege: Public

Associations:

The DsDdDataItem class has associations with the following classes:
DsDdDistList (Aggregation)

7.3.4 DsDdDistFile Class

Parent Class: DsDdDataItem
Public: YesDistributed Object: No
Persistent Class: True
Purpose and Description:
File containing data to be distributed to the requestor.

Attributes:

myPath - Specification of where the file is located, including device and directory.
Data Type: char *
Privilege: Private
Default Value:

Operations:

Deobjectize - Returns as a C++ struct the data stored by the class. This is necessary for the data to be transported via OODCE, since OODCE does not support the transport of objects as such.
Arguments:
Return Type: DistFile
Privilege: Public

Associations:

The DsDdDistFile class has associations with the following classes:
None

7.3.5 DsDdDistList Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class: True
Purpose and Description:
List of pointers to all the items to be distributed. This class is derived from a RogueWave template, and so inherits all of the operations (not shown) to manipulate the list.

Attributes:

None

Operations:

Deobjectize - Returns as a C++ struct the data stored by the class. This is necessary for transport of the data via OODCE, since OODCE does not transport objects as such.
Arguments:
Return Type: DistList
Privilege: Public

Distribute - Initiates distribution processing for all of the items in the list. This signature definition supports all requests except for electronic push requests.
Arguments: Media:char *, User:MsUsProfile, Format:char *,
Return Type: DsDdDistRequest
Privilege: Public

Distribute - Initiates distribution processing for items to be distributed via electronic push.
Arguments: Media:char *, User:MsUsProfile, ElecDestn:char *, Password:char *, Format:char *
Return Type: DsDdDistRequest
Privilege: Public

Associations:

The DsDdDistList class has associations with the following classes:
Class: DsDdDistRequest requestsdistributionofitemsin - A distribution request is the vehicle for performing the distribution of the items in the distribution list.

7.3.6 DsDdDistRequest Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
Base class for the client/server specialization of distribution requests. Specifies operations which are available to any client.

Attributes:

None

Operations:

Abort - Terminate all processing of a distribution request. The termination may not occur immediately, because the request may need to progress to a stable stage enabling clean termination.
Arguments:
Return Type: EcStatus
Privilege: Public
PDL:// if (thread which called DsStResource.Allocate is blocked because resource is not yet available)
{
call DsStRequestManager.Abort to abort the resource allocation
}
else
}
call SetAbortFlag to signal to ServiceRequest to abort processing

}

GetState - Get the state of the distribution request. States defined in the L4 requirements are: pending, active, waiting for shipment, shipped. An additional state of suspended is possible.
Arguments:
Return Type: DistState
Privilege: Public

Submit - Submit a distribution request for processing. This definition of the submit signature supports all distribution except electronic push, which requires additional arguments.
Arguments: Media:char *, User:MsUsProfile, Format:char *
Return Type: Void
Privilege: Private

Submit - Submit a distribution request for processing. This definition of the submit signature supports electronic push only.
Arguments: Media:char *, User:MsUsProfile, ElecDestn:char *, Password:char *, Format:char *
Return Type: Void
Privilege: Private

WaitForComplete - Wait until a distribution request has completed. Since the submission of a distribution request is asynchronous, this service allows a thread to block until (and thereby be signalled when) the requests completes.
Arguments:
Return Type: EcStatus
Privilege: Public

Associations:

The DsDdDistRequest class has associations with the following classes:
Class: DsDdDistList requestsdistributionofitemsin - A distribution request is the vehicle for performing the distribution of the items in the distribution list.

7.3.7 DsDdDistRequestC Class

Parent Class: DsDdDistRequest
Public: YesDistributed Object: Yes
Purpose and Description:
User client presentation of the distribution request. A specialization of the constructor is

necessary to accomodate hiding of the request manager (which implements the OODCE factory model) from the client.

Attributes:

All Attributes inherited from parent class

Operations:

DsDdDistRequestC - Constructor for the user client presentation of the distribution request. This constructor implements the client side of the OODCE factory model; it first creates a client factory object if it doesn't exist, then calls the appropriate factory service to create a server-side distribution request.
Arguments:
Return Type: DCEObjRefT
Privilege: Public
PDL:// call DsDdDistList.Deobjectize to flatten the list for transfer to the server via OODCE

```
// if ( client representation of request manager - aka the OODCE factory - doesn't exist )
{
  // construct DsDdRequestManagerC object
}
```

// call DsDdRequestManagerC.CreateDistRequest to create server-side representation of the distribution request

// call OODCE's SetBinding to bind the client distribution request object to the server object reference returned by CreateDistRequest

// call DsDdDistRequest.Submit to initiate processing of the request.

Associations:

The DsDdDistRequestC class has associations with the following classes:
None

7.3.8 DsDdDistRequestS Class

Parent Class: DsDdPrivRequest
Public: NoDistributed Object: Yes
Persistent Class: True

Purpose and Description:
Server-side presentation of the distribution request. This side has all the services available to all of the clients, and attributes which are maintained as such only on the server side. An instantiation of this class exists for each distribution request in the system, and runs in a separate thread which is created by the request factory (manager).

Attributes:

myAbortFlag - Set by a thread executing the Abort service, signals to a thread executing the ServiceRequest service that the request is to be aborted.
Data Type: RWBoolean
Privilege: Private
Default Value: False

myDistFormat - Format - tar, cpio, or none - in which to write the distribution data to the output media.
Data Type: char *
Privilege: Private
Default Value:

myDistList - List of items to be distributed.
Data Type: DsDdDistList *
Privilege: Private
Default Value:

myMedia - Type of media to be be used for the distribution. For release A possible values are CD, 8mm tape, electronic push, electronic pull.
Data Type: char *
Privilege: Private
Default Value:

myPriority - Priority at which the distribution request is processed relative to other distribution requests.
Data Type: DistPriority
Privilege: Private
Default Value: medium

myRequestId - Unique identifier for the request, which is written with any event log messages and so can be used to reconstruct the events that occured for the given request. The request id is of minimal value while a request is active, because all client software interfaces to the request should be via of the request object; however, once a request completes the object disappears, so the the request id can then be used to trace through the request's audit trail.
Data Type: char *

Privilege: Private
Default Value:

myRequestProcessor - Specialized request processor whose ServiceRequest service will be invoked to generate the distribution products. The value for this attribute is assigned based upon the media type; e.g., for a media type of CD the DsDdCDProcessor will be used.
Data Type: DsDdRequestProcessor *
Privilege: Private
Default Value:

mySize - Total size of the data to be distributed in the request. Due to addressing limits (of 32-bit architecture), the units for this count may not be bytes, but may be some TBD unit such as kilobytes.
Data Type: int
Privilege: Private
Default Value:

myState - Request states identified in the level 4 requirements are pending, active, waiting for shipment, and shipped. An additional state of suspended is added to support the ability to suspend a request. Pending request are requests which have been received but for which a resource (media) to which to write the data is not yet available. The state changes to active when the resource becomes available, and remains active until the generation of all output media is complete, at which time the state changes to waiting to be shipped. The state changes to shipped upon operator notification to the software that the media have been physically shipped to the requestor.
Data Type: DistState
Privilege: Private
Default Value: pending

myTimeOfSubmission - Time when the Submit service was invoked upon the request.
Data Type: RWDate
Privilege: Private
Default Value:

myUserInfo - Information identifying the entity (user, DAAC, etc.) which requested the distribution.
Data Type: MsUsProfile *
Privilege: Private
Default Value:

Operations:

All Operations inherited from parent class

Associations:

The DsDdDistRequestS class has associations with the following classes:
Class: DsDdRequestManagerS creates
DsDdRequestList (Aggregation)

7.3.9 DsDdElectronicMedia Class

Parent Class: DsDdMedia
Public: NoDistributed Object: No
Purpose and Description:
Handles the interface to storage management for distribution which utilize network communications for the distributions of the files. No physical shipment of the data items selected is made so there is no need for media or shipping labels.

Attributes:

myCapacity - Capacity for holding data: size of a single tape or CD or the amount of staging available for this request.
Data Type: int
Privilege: Private
Default Value:

mySourceDirectory - Directory local to distribution processing in which a tar or cpio file will be created if necessary for the request.
Data Type: char*
Privilege: Private
Default Value:

Operations:

CreateCpioFile - Creates a Cpio archive file on local staging disk. The file will then be distributed via electronic push or pull.
Arguments:
Return Type: Void
Privilege: Private

CreateOutputFile - Determines if a tar or cpio archive file needs to be created for this request based on the distribution format passed in.
Arguments: DistFmt:char*
Return Type: FileName:char*
Privilege: Private
PDL://switch DistFmt

```
  {
//    tar:
//        FileName = CreateTarFile() create a tar archive file
//        in local storage
//    cpio:
//        FileName = CreateCpioFile() create a cpio archive file
//        in local storage
//    raw:
//        FileName = NULL
//    default:
//        signal an error
  }
```

CreateTarFile - Creates a tar archive file of the data items. The file is then distributed via electronic push or pull.
Arguments:
Return Type: Void
Privilege: Private

DsDdElectronicMedia - Constructor for the Electronic Media instance.
Arguments:
Return Type: Void
Privilege: Public

~DsDdElectronicMedia - Destructor for an instance of the Electronic Media class
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdElectronicMedia class has associations with the following classes:
None

7.3.10 DsDdLabeledMedia Class

Parent Class: DsDdMedia
Public: NoDistributed Object: No
Purpose and Description:
This object handles the transfer of data item files to labeled media such as CD and tape. It is specialized for the operation of printing the label which may have different formats for the different resources.

Attributes:

myCapacity - Size of a single tape (accounting for density) or single CD.
Data Type: int
Privilege: Private
Default Value:

myDesignatedPrinter - Holds the name of the printer designated for printing labels for media. This will not be needed until Release B.
Data Type: char *
Privilege: Private
Default Value:

myNumVolumes - The number of volumes of removable media that were created for this request.
Data Type: int
Privilege: Private
Default Value:

myShippingLabel - Reference to the shipping label object which will create the label conataining the requestor's name and shipping address for use in the shipment of the media.
Data Type: DsDdShippingLabel &
Privilege: Private
Default Value:

Operations:

CopyFiles - Creates and executes a shell command file to move the requested data items into an archive file of the requested type on the requested media.
Arguments: DistFmt:char*
Return Type: Void
Privilege: Private
PDL://ShellCmd = CreateShellCmd(DistributionFormat) to create a shell

```
//command which will create the requested type of archive file
// on the requested type of media

//if ShellCmd is not NULL
  {
//    Execute the shell command to move the files to the allocated
//    media resource

//    DetermineNumVols() to determine the number of volumes created
//    for this distribution

//    DsDdPackingSlip::SetNumVolumes(myNumVolumes) to set the
//    number of volumes in the Packing Slip object
  }
```

CreateCpioCmd - Creates a shell command which will create a cpio archive file on the requested media. The archive file will contain all of the data items listed in the list file.
Arguments:
Return Type: ShellCmd:char*
Privilege: Private

CreateShellCmd - Creates a shell command which will create a tar or cpio archive file on the requested media.
Arguments: DistFmt:char*
Return Type: ShellCmd:char*
Privilege: Private

```
PDL://CreateFileList() to create a file containing  the data items and
//their file paths.

//DsStResource::GetResourceId() to get the name of the device which
//has been allocated for this request.

//switch(DistributionFormat)
  {
//   tar:
//       ShellCmd = CreateTarCmd() use the name of the resource and
//       the name of the file containing the list of data items
//       to create a command which will create a tar archive file
//       on the allocated device
//    cpio:
//       ShellCmd = CreateCpioCmd() uses the name of the resource
//       and the name of the file containing a list of the data
//       items to create a  command which will create a cpio
//       archive file on the allocated device
//     default:
```

```
//      ShellCmd = NULL
  }
```

CreateTarCmd - Creates a shell command which will create a tar archive file of the data items listed in the list file on the requested media.
Arguments:
Return Type: ShellCmd:char*
Privilege: Private

DetermineNumVols - A service to determine the number of volumes which were written for a distribution using removable physical media.
Arguments:
Return Type: int
Privilege: Private

DsDdLabeledMedia - Constructor for the Media Label class instance.
Arguments: ShippingLabel:DsDdShippingLabel &
Return Type: Void
Privilege: Public

PrintMediaLabel - Prints the label whose format will be specific for different media on the designated printer. This will be implemented in release B.
Arguments:
Return Type: Void
Privilege: Public
This is an abstract operation

TransferFiles - Concrete implementation of the interface with the Request Processor object. This method handles the transfer of files to a tar or cpio archive file on the requested media.
Arguments: DistFmt:char*
Return Type: Void
Privilege: Public
PDL:

```
//DsStResourceC::Allocate(ResourceType,MediaType,RequestId,
//Priority,Size,ProfileInfo) to allocate a resource of the
//requested type.

//CopyFiles(DistFmt) to copy the files to the requested device

//Call DsStResourceC::Deallocate() to free the requested device
//for other use

//DsDdPackingSlip::Create() to create the packing slip.  This is
//done after the completion of the physical distribution to allow
```

//inclusion of the number of volumes created.

//DsDdPackingSlip::Print() to send the packing slip to the
//designated printer

~DsDdLabeledMedia - Destructor for a labeled media instance.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdLabeledMedia class has associations with the following classes:
None

7.3.11 DsDdMedia Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To track the content of distribution media produced by the Data server. Physical distribution media currently identified for distribution includes tapes (3480/3490, 4mm 8mm,6250) and CD ROM disks. The contents of media produced for distribution will be reflected by a packing slip which will be included in the shipping container.

Attributes:

myDateCreated - Date that the file transfer to the requested media was initiated.
Data Type: GlDate
Privilege: Private
Default Value:

myDistributionList - Reference to the DsDdDistList object which is associated with this request. Used to allow access to the list of data items for the request.
Data Type: DsDdDistList &
Privilege: Private
Default Value:

myFileListPtr
Data Type: char *
Privilege: Private

Default Value:

myMediaType - Holds the type of media, i.e. tape, CDROM, network, source and associated information.
Data Type: MediaType
Privilege: Private
Default Value:

myNumberOfItems - Number of Data Items distributed for this request.
Data Type: int
Privilege: Private
Default Value:

myPackingSlip - Reference to the DsDdPackingSlip object which will create the list of data items which were transfered in this distribution.
Data Type: DsDdPackingSlip
Privilege: Private
Default Value:

myRequest - Reference to the DsDdRequestProcessor object which represents this request. Allows access to the information in the request which is not part of the media object.
Data Type: DsDdRequestProcessor &
Privilege: Private
Default Value:

myResource - The name of the resource which is allocate for the transfer of the files.
Data Type: DsStResourceC
Privilege: Private
Default Value:

Operations:

CreateFileLIst - Creates a file with the list of all the files and their path names for use in constructing the commands which will transfer the files and building the packing slip.
Arguments:
Return Type: Void
Privilege: Private

DsDdMedia
Arguments: DistributionList:DsDdDistList &, Request:DsDdRequestProcessor &, PackingSlip:DsDdPackingSlip &, Resource:DsStResourceC &, MediaType:MediaType

TransferFiles - Interface routine to the Request Processor object. This operation will have different methods depending on the media type requested . For this class, it is a pure virtual

operation.
Arguments: DistFmt:char*
Return Type: Void
Privilege: Public
This is an abstract operation

~DsDdMedia - Destructor for the media object. Frees up all allocated memory used in constructing the object.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdMedia class has associations with the following classes:
Class: DsDdRequestProcessor DataItemsTransferedBy
Class: DsDdPackingSlip GetsTransferInfoFrom
Class: DsDdDistList GivesDataItemInfoTo
Class: DsStResourceC TransfersDataItemsTo

7.3.12 DsDdOpsRequestC Class

Parent Class: DsDdPrivRequest
Public: YesDistributed Object: Yes
Purpose and Description:
Presentation of the distribution request services to the operator client. This client has access to privileged services which are not available to the user client.

Attributes:

All Attributes inherited from parent class

Operations:

DsDdOpsRequestC - The constructor for the operator client request interfaces to the request factory (manager) for creation of the server-side request.
Arguments:
Return Type: DCEObjRefT
Privilege: Public

Associations:

The DsDdOpsRequestC class has associations with the following classes:
None

7.3.13 DsDdPackingSlip Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:
This object serves as the list of items that were successfully transfered for a given distribution request.

Attributes:

myDesignatedPrinter - Name of the printer designated for printing the packing list for shipment of removable media.
Data Type: char*
Privilege: Private
Default Value:

myFileName - Name of the file containing the packing slip. The file will either be printed for physical shipment, sent electronically with electronic distribution, or handled in some other manner for future distribution options.
Data Type: FileName
Privilege: Private
Default Value:

myItemList - Reference to the distribution list for this distribution. The distribution list is read item by item to create a file which contains a list of all the data items distributed.
Data Type: DsDdDistList &
Privilege: Private
Default Value:

myNumVolumes - The number of volumes for physical distribution.
Data Type: int
Privilege: Private
Default Value: 0

Operations:

CreatePackSlip - Creates a file with a list of the data items that were included in this distribution request.
Arguments:
Return Type: GlStatus
Privilege: Public

DsDdPackingSlip - Constructor for the packing slip instance.
Arguments: ItemList:DsDdDistList&
Return Type: Void
Privilege: Public

Print - Prints the packing slip to the printer designated for packing slips
Arguments:
Return Type: GlStatus
Privilege: Public

~DsDdPackingSlip - Destructor for the packing slip instance. The flle used to hold the packing slip will be removed when the instance is removed.
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdPackingSlip class has associations with the following classes:
Class: DsDdMedia GetsTransferInfoFrom
Class: DsDdRequestProcessor ListsItemsDistributedBy
Class: DsDdDistList listsitemsin

7.3.14 DsDdPrivRequest Class

Parent Class: DsDdDistRequest
Public: YesDistributed Object: No
Purpose and Description:
Specialization of the distribution request to provide access to privileged operations. Privileged operations are available to operator clients but not user clients.

Attributes:

All Attributes inherited from parent class

Operations:

RestartOutput - Restart the output to media of a distribution request. This service is provide to support the operator restarting media output in the event of a failure, such as fatal tape errors during writing of a tape.
Arguments:
Return Type: EcStatus
Privilege: Public

Resume - Resume processing of a distribution request for which processing was suspended.
Arguments:
Return Type: EcStatus
Privilege: Public

SetPriority - Changes the priority of a distribution request, which the operator may need to do when managing the flow of distribution requests.
Arguments: Priority:DistPriority
Return Type: Void
Privilege: Public

Suspend - Suspend the processing of a distribution request. Suspension may not occur immediately, because the request processing may need to proceed to a stable point which enables processing of the suspended request to be cleanly resumed when desired.
Arguments:
Return Type: EcStatus
Privilege: Public

Associations:

The DsDdPrivRequest class has associations with the following classes:
None

7.3.15 DsDdPullMedia Class

Parent Class: DsDdElectronicMedia
Public: NoDistributed Object: No
Purpose and Description:
Initiates and monitors the distribution of data items to the pull staging area

Attributes:

All Attributes inherited from parent class

Operations:

DsDdPullMedia - Object constructor
Arguments:
Return Type: Void
Privilege: Public

GetPackingListFile - Returns the file name from the packing slip object. Its primary use is to allow inclusion of the packing slip in the notification to the user for some distributions (e.g. pull).
Arguments:
Return Type: FileName
Privilege: Public

PullFiles - Handles the interface with storage management to move the requested data items to staging disk which is set aside for pull distribution.
Arguments:
Return Type: Void
Privilege: Private
PDL://DsDdPackingSlip::CreatePackSlip() to creaete the list of data
//items in this distribution

//DsStResourceC::Allocate(ResourceType,MediaType,RequestId,
//Priority,Size,ProfileInfo) to allocate storage for the Pull
//request

//while there are items to be distributed
{
// DataItem = DsDdDistList::GetNextItem() get then next
// data item for distribution

// Parse Data Item location information to get node and path

```
//    components

//    DsStResourceC::DistFrom(DataItemNode,"PULL",DataItemPath) to
//    move that data item to the pull storagae area
   }

// DsStResourceC::Deallocate() to free up the resource for other
// uses
```

PullFmtFiles
Arguments: FileName:FileName
Return Type: Void
Privilege: Private

TransferFiles - Mediates the transfer of data items to the pull storage area. If needed, a tar or cpio archive file is created.
Arguments: DistFmt:char*
Return Type: Void
Privilege: Public

```
PDL://FileName = CreateOutputFile(DistFmt) create a tar or cpio file
//containing the data items if needed.  If no tar/cpio file has
//been requested, FileNmae will be Null

//if FileName is NULL
  {
//    PullFiles()
  }
//else
  {
//    PullFmtFiles(FileName)
  }
```

~DsDdPullMedia - Object destructor
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdPullMedia class has associations with the following classes:
None

7.3.16 DsDdPullProcessor Class

Parent Class: DsDdRequestProcessor
Public: NoDistributed Object: No
Purpose and Description:
Provides concrete instance of the CreateMedia which processes an electronic pull request.

Attributes:

All Attributes inherited from parent class

Operations:

CreateMedia - Creates the instances of the objects which will interface with the devices needed for the transfer of data items associcated with this request.
Arguments:
Return Type: Void
Privilege: Private

CreateNotificationMsg
Arguments:
Return Type: RWCString
Privilege: Public

Associations:

The DsDdPullProcessor class has associations with the following classes:
None

7.3.17 DsDdPushMedia Class

Parent Class: DsDdElectronicMedia
Public: NoDistributed Object: No
Purpose and Description:
Initiates and monitors the push of DataItems or tar/cpio archive files consisting of DataItems across the network to the requestor.

Attributes:

myDirectory - Directory on the remote node where files will be copied by the push distribution
Data Type: EcTChar*
Privilege: Private
Default Value:

myLogin - login information supplied by the user: an account for the node and the password for that account.
Data Type: Login
Privilege: Private
Default Value:

myRemoteNode - Indicates the node to which data items or tar/cpio archive files of data items will be pushed
Data Type: EcTChar*
Privilege: Private
Default Value:

Operations:

DsDdPushMedia - Constructor for the push media object.
Arguments: Node:EcTChar*, Login:Login, Directory:EcTChar*
Return Type: Void
Privilege: Public

PushFiles - Distributes the requested data items without any format change to the node and directory supplied by the request using the account and password also supplied by the request.
Arguments:
Return Type: Void
Privilege: Private
PDL://DsDdPackingSlip::CreatePackSlip() create a file with a list of
//the data items which will be pushed to the requestor

//DsStResourceC.Allocate(ResourceType,MediaType,RequestId,Priority,
//Size,ProfileInfo) to allocate a network resource for "pushing"
//the data items to the requestor

//DsStResourceC::DistFrom(DistributionNode,"PUSH",PackSlip) to
//send the packing slip to the requestor

//while there are files to be distributed
{

```
//   DataItem = DsDdDistList.GetNextItem() to get the next data
//   item for distribution

//   Parse data item location to get node and path of the data
//   item

//   DsStResourceC::DistFrom(DataItemNode,"PUSH",DataItemPath)
//   to push the data itme over the network resource
  }

//DsStResourceC::Deallocate() to free the resource for other use
```

PushFmtFile - Creates a tar or cpio archvie file containing the data items. The tar/cpio archive file is then pushed to the requestor along with the packing slip (a list of data items that should be in the tar/cpio archive file).
Arguments: FileName:char*
Return Type: Void
Privilege: Public

```
PDL://DsDdPackingSlip::CreatePackSlip() create a file with the list of
//data items included in this distribution

//DsStResourceC::Allocate(RequestType,MediaType,RequestId,Priority,
//Size,ProfileInfo) to obtain a network resource for the push

//DsStResourceC::DistFrom(DistributionNode,"PUSH",PackingSlip) to
//transfer the packing slip to the node and directory specified
//by the requestor

//DsStResourceC::DistFrom(DistributionNode,"PUSH",FileName) to
//transfer the tar/cpio archive file which contains the requested
//data items

//DsStResourceC::Deallocate() to free up the resource for other use

//Remove the tar/cpio archive file from local storage
```

TransferFiles - Concrete specialization of the interface with the RequestProcessor object. Creates a tar/cpio archive file if needed and sends the data items to the reqeustor.
Arguments: DistFmt:char*
Return Type: Void
Privilege: Public

```
PDL://FileName = CreateOutputFile(DistFmt) create a tar/cpio archive
//file if needed based on the distribution format submitted in the
//request
```

```
//if FileName is NULL - no tar or cpio archive was created
  {
//   PushFiles() to push the files one at a time to the requestor
  }
//else  - data items have been put in a tar/cpio archive file
  {
//   PushFmtFile(FileName) to push a tar/cpio archive file to
//   the requestor
  }
```

~DsDdPushMedia - Destructor for the object
Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdPushMedia class has associations with the following classes:
None

7.3.18 DsDdPushProcessor Class

Parent Class: DsDdRequestProcessor
Public: NoDistributed Object: No
Purpose and Description:
Defines the concrete CreateMedia which performs processing for electronic push distribution.

Attributes:

myDestPassword - Password to use at the destination node.
Data Type: char
Privilege: Private
Default Value:

myDestination - Destination for the electronic push, which includes destination TCP/IP address and directory path.
Data Type: char
Privilege: Private
Default Value:

myLogin - Login name to use at the destination node.

Data Type: char
Privilege: Private
Default Value:

Operations:

CreateMedia - Creates the instances of the objects which will interface with the devices needed for the transfer of data items for this request.
Arguments:
Return Type: Void
Privilege: Private

Associations:

The DsDdPushProcessor class has associations with the following classes:
None

7.3.19 DsDdRequestList Class

Parent Class: Not Applicable
Public: YesDistributed Object: No
Persistent Class: True
Purpose and Description:
Set of pointers to all distribution requests.

Attributes:

None

Operations:

Sort - Sort the requests by a particular attribute. Level 4 requirements exist to display requests by request id, state, or request type (electronic or physical media).
Arguments: Key:SortTypes
Return Type: DsDdRequestList
Privilege: Public

Associations:

The DsDdRequestList class has associations with the following classes:
None

7.3.20 DsDdRequestManager Class

Parent Class: Not Applicable
Public: YesDistributed Object: Yes
Purpose and Description:
Base class for implementation of the OODCE factory model. Derived classes manufacture server-side distribution requests and provide client-side presentations of those requests.

Attributes:

None

Operations:

CreateDistRequest - Creates a new distribution request. Two definitions of this service exists; this definition creates a distribution request for all requests except for electronic push requests, which require additional arguments.
Arguments: List:DistList, Media:char *, User:MsUsProfile, Format:char *
Return Type: DCEObjRefT
Privilege: Public
PDL:// construct server-side DsDdDistFile object
// construct server-side DsDdDistList object
// call DsDdDistList.Insert to place file object in the list

// construct DsDdDistRequestS
// call OODCE's RegisterObject to make the distribution request object visible as a distributed object
// call and return DCE's GetObjectReference to provide the distributed object reference to the client

CreateDistRequest -
Arguments: List:DistList, Media:char *, User:MsUsProfile, ElecDestntn:char *, Password:char *, Format:char *
Return Type: DCEObjRefT
Privilege: Public

DeleteDistRequest - Service, in the OODCE factory model, to delete a distribution request

which was created via the CreateDistRequest service.
Arguments: id:uuid_t
Return Type: Void
Privilege: Public

InventoryRequests - Provides an inventory - in the form of a list of distributed object references - of all distribution requests.
Arguments:
Return Type: DsDdRequestList
Privilege: Public

Associations:

The DsDdRequestManager class has associations with the following classes:
None

7.3.21 DsDdRequestManagerC Class

Parent Class: DsDdRequestManager
Public: YesDistributed Object: Yes
Purpose and Description:
Client-side presentation of the request factory. The client's applications code will be unaware of the existence of this object, because this object will be created and managed from within the client instance of the distribution request. While this may seem to be a chicken and egg paradox - the distribution request creates the factory, which creates the distribution request - is isn't, because the logic is actually: 1- the client-side distribution request is created, which 2- creates the factory, which 3- is used to create a server-side instance of the distribution request, 4- whose OODCE distributed reference is returned, to the client caller, as the client-side distribution request

Attributes:

All Attributes inherited from parent class

Operations:

All Operations inherited from parent class

Associations:

The DsDdRequestManagerC class has associations with the following classes:
None

7.3.22 DsDdRequestManagerS Class

Parent Class: DsDdRequestManager
Public: NoDistributed Object: Yes
Purpose and Description:
Server-side presentation of the Distribution Request Factory. Manufactures Distribution Requests, the distributed reference for which is returned to the client.

Attributes:

myRequestList - Pointer to list of all current distribution requests.
Data Type: DsDdRequestList
Privilege: Private
Default Value:

Operations:

All Operations inherited from parent class

Associations:

The DsDdRequestManagerS class has associations with the following classes:
Class: DsDdDistRequestS creates

7.3.23 DsDdRequestProcessor Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Purpose and Description:
This base class provides to the distribution request a specialized class with the correct ServiceRequest operation to perform the processing for the media specified in the distribution request. This class and its specializations are a hybrid of the Template and Strategy patterns described in the book "Design Patterns" by Gamma et al.

Attributes:

myRequest - The request processor points back to the distribution request so it can access request attributes. The request process will be designated as a friend of the distribution request class.
Data Type: DsDdDistRequestS
Privilege: Private
Default Value:

Operations:

CreateMedia - Creates the instances of the objects needed to transfer the data items to the requested media in the requested format.
Arguments:
Return Type: Void
Privilege: Private
This is an abstract operation

CreateNotificationMsg
Arguments:
Return Type: RWCString
Privilege: Private

DsDdRequestProcessor - Two definitions of the constructor exist; this definition handles all requests except those for electronic push. Based upon the media type, this constructor will create the appropriate specialized (DsDd..Processor) object.
Arguments: Media:MediaType, Request:DsDdDistRequestS
Return Type: Void
Privilege: Public

DsDdRequestProcessor - This definition contains the arguments needed for construction of a DsDdPushProcessor.
Arguments: Media:MediaType, Request:DsDdDistRequestS, ElecDestination:char, Password:char
Return Type: Void
Privilege: Public

ServiceRequest - Creates the auxillary objects needed to complete the request and interfaces with external logging and notification services. This service is inhereted by all of the subclasses and should not be overridden.
Arguments:
Return Type: GlStatus
Privilege: Public
PDL://

EcUtLogger::Log(EventNumber,EventType,Disposition,Time,Message,[parameters
// for message]requestor,ReceiptTime,MediaType,RequestSize,Destination) to log
// the receipt of the message

//CreateMedia() create the objects needed for the transfer of the data items
// to the requested media

//myMedia->TransferFiles(DistFmt:char*) delegate the transfer of files to the
//created media object

//EcUtLogger::Log(EventNumber,EventType,Disposition,Time,Message,[parameters
//for message]requestor,CompletionTime,NumberofVolumes,NumberofItems,
//TotalSize,Destination) to log the completion of the request

//CreateNotificationMsg() creates a message to the requestor with the needed
//information

//GlNotification::SenndToUser(Message) to notify the requestor of the
//completion of the request

Associations:

The DsDdRequestProcessor class has associations with the following classes:
Class: DsStResourceC Allocates
Class: DsDdMedia DataItemsTransferedBy
Class: DsDdPackingSlip ListsItemsDistributedBy
Class: EcUtLogger LogsEventsVia
Class: DsDdShippingLabel Usesformailingphysicalmedia
Class: EcNotification notifiesuservia
DsDdDistRequestS (Aggregation)

7.3.24 DsDdShippingLabel Class

Parent Class: Not Applicable
Public: NoDistributed Object: No
Persistent Class:
Purpose and Description:
To contain information indicating the shipping destination for distribution media. A media label is produced for each shipment of distribution data on physical media.

Attributes:

myDesignatedPrinter - Printer designated for printing the shipping label.
Data Type: char*
Privilege: Private
Default Value:

myFileLocation - Name of the file (including path) which contains the shipping Label.
Data Type: char*
Privilege: Private
Default Value:

myUserInfo - Reference to the object containing the profile of the requestor. Name and shipping address of the requestor are needed.
Data Type: MsUsProfile &
Privilege: Private
Default Value:

Operations:

CreateLabel - Creates a shipping label using the user profile information from the request. The shippping label consists of the requestor's name and shipping address and will be used to address the physical shipment to the requestor. Since the required information for the shipping label is present when the label object is created, the constructor for the object calls this service and creates a file containing the shipping label.
Arguments:
Return Type: Void
Privilege: Private

DsDdShippingLabel - Constructor for the shipping label. Since all information needed for the shipping label is present when the constructor is activated, the constructor creates a file holding the shipping label.
Arguments: UserInfo:MsUsProfile &
Return Type: Void
Privilege: Public

PrintShipLabel - Sends the shipping label to the designated printer which is assumed to have loaded the forms for the shipping label.
Arguments:
Return Type: GlStatus
Privilege: Public

~DsDdShippingLabel - Destructor of an instance of the shipping label. This operation removes the file containing the shipping label.

Arguments:
Return Type: Void
Privilege: Public

Associations:

The DsDdShippingLabel class has associations with the following classes:
Class: DsDdRequestProcessor Usesformailingphysicalmedia
Class: MsUsProfile isusedin

7.3.25 DsDdTapeMedia Class

Parent Class: DsDdLabeledMedia
Public: NoDistributed Object: No
Purpose and Description:
Handles the initiation and monitor of the transfer of data items to magnetic tape as part of a tar or cpio archive file

Attributes:

myDensity - Requested density for the write to tape. Values are high, medium, and low
Data Type: char*
Privilege: Private
Default Value: "low"

Operations:

DsDdTapeMedia - Constructor for an instance of the Tape Media object.
Arguments: Density:char*
Return Type: Void
Privilege: Public

PrintMediaLabel - For release B implementation. Will handle the differences in label contents and formats between tape and cd
Arguments:
Return Type: Void
Privilege: Public

~DsDdTapeMedia - Destructor for an instance of the Tape Media object.
Arguments:

Return Type: Void
Privilege: Public

Associations:

The DsDdTapeMedia class has associations with the following classes:
None

7.3.26 DsDdTapeProcessor Class

Parent Class: DsDdRequestProcessor
Public: NoDistributed Object: No
Purpose and Description:
Defines the concrete CreateMedia which performs the processing for tape media.

Attributes:

myDensity
Data Type: char *
Privilege: Private
Default Value: "low"

Operations:

CreateMedia - Creates instances of the objectes which will interface with the devices needed for transfer of the data items associated with this request.
Arguments:
Return Type: Void
Privilege: Protected

Associations:

The DsDdTapeProcessor class has associations with the following classes:
None

7.3.27 DsStResourceC Class

Parent Class: Not Applicable

Attributes:

None

Operations:

None

Associations:

The DsStResourceC class has associations with the following classes:
Class: DsDdRequestProcessor Allocates
Class: DsDdMedia TransfersDataItemsTo

7.3.28 EcNotification Class

Parent Class: Not Applicable

Attributes:

None

Operations:

None

Associations:

The EcNotification class has associations with the following classes:
Class: DsDdRequestProcessor notifiesuservia

7.3.29 EcUtLogger Class

Parent Class: Not Applicable

Attributes:

None

Operations:

None

Associations:

The EcUtLogger class has associations with the following classes:
Class: DsDdRequestProcessor LogsEventsVia

7.3.30 MsUsProfile Class

Parent Class: Not Applicable

Attributes:

None

Operations:

None

Associations:

The MsUsProfile class has associations with the following classes:
Class: DsDdDistRequestS isforuseridentifiedvia
Class: DsDdShippingLabel isusedin

7.4 CSCI Dynamic Model

This section describes several scenarios for the Data Distribution CSCI. A scenario delineates the steps that occur in a particular execution through the system.

Since all actions are initiated via a distribution request which may be a distributed object, the first scenario (Distributed Creation of a New Distribution Request) presents both the client- and server-side events involved in the creation of a distribution request distributed object. The sequence of events depicted in this scenario are unvarying for all additional scenarios involving the creation of a new request. Since the client/server specializations of the request perform logically as a single (albeit distributed) object, in subsequent scenarios the client-/server-side representation is omitted, and the distribution request distributed object is treated as single object.

The scenarios presented in this section are:

- Distributed Creation of a New Distribution Request

- Electronic Pull of Data
- Electronic Push of Data
- Physical Distribution to Tape
- Abort of an Active Request
- Tape Fault

7.4.1 Distributed Creation of a New Distribution Request

Summary

Figure 7.4.1-1 depicts the creation of a new distribution request. The client (DsSrWorkingCollection) creates one or more objects of class DsDdDistFile, inserts them into a DsDdDistList, and invokes the Distribute service. The Distribute service will create a new distributed object DsDdDistRequest, which is created on the server via the OODCE factory model implemented in DsDdRequestManager. The distributed creation of DsDdDistRequest also results in the deobjectization of the client DsDdDistFile and DsDdDistList and their reconstruction on the server side, which is necessary because OODCE does not provide a deep copy of referenced objects.

In the remaining event traces the client/server distinction of classes and events presented in this diagram will be depicted at a higher level which does not distinguish between client and server events or classes; for instance, DsDdDistRequest will be shown instead of showing both DsDdDistRequestC and DsDdDistRequestS. Additionally DsDdRequestManager, which implements the OODCE factory model and is transparent to the client and ancillary to the main event flow, will not be shown on subsequent traces.

Assumptions

- The user desktop has an active dataserver session.
- The job's working collection is populated with the correct ESDTs.
- The distribution interface is up on the client's desktop

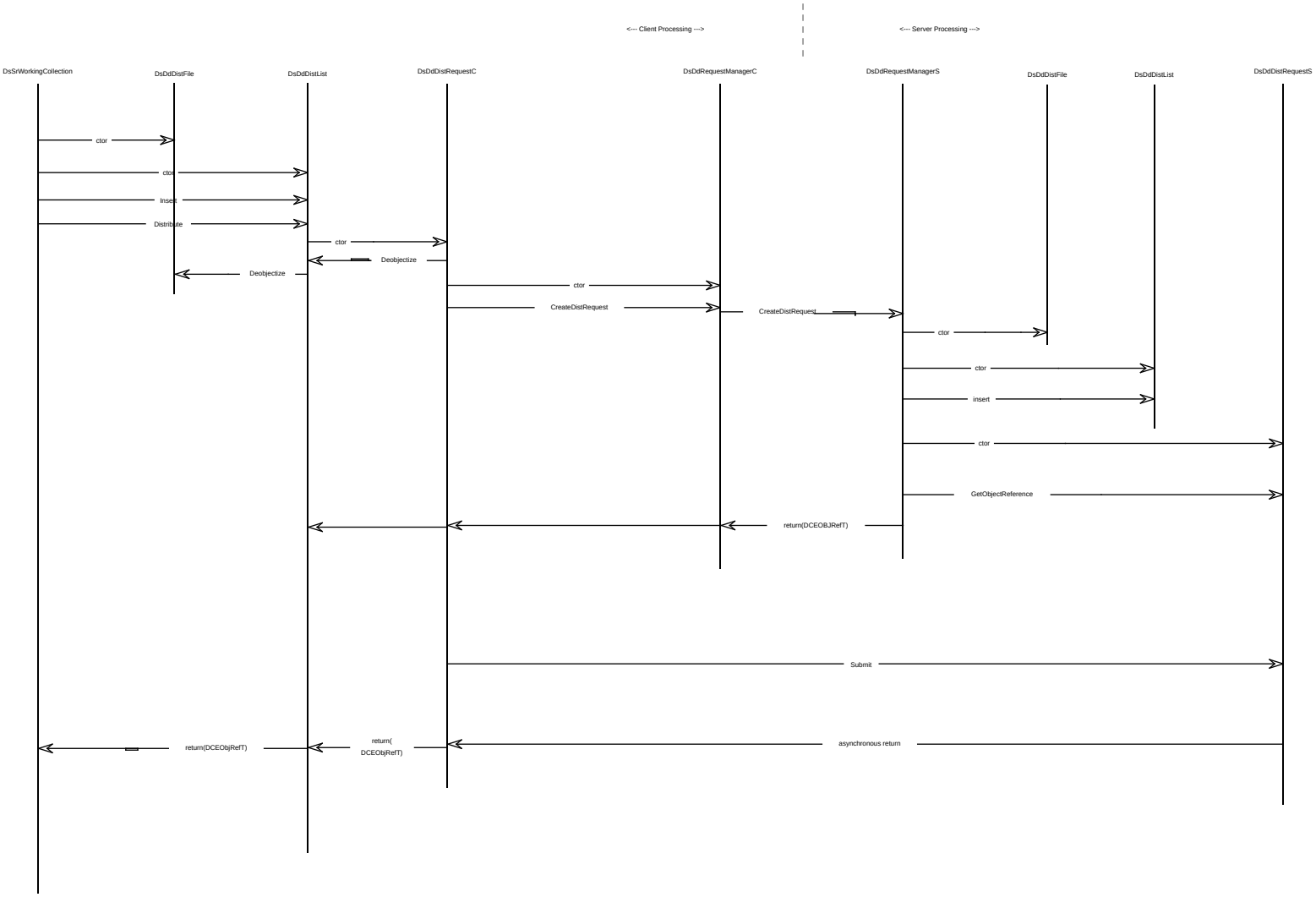


Figure 7.4.1-1. Creating a Distribution Request

7.4.2 Electronic Pull of Data

Summary

Figure 7.4.2-1 depicts the distribution of data via electronic pull. The scenario begins with DsSrWorkingCollection invoking the Distribute service, which results in the creation and asynchronous return of a DsDdDistRequest, followed by the submission of that request. The request is added to the DsDdRequestList (a list of all current requests), and the media type (in this case electronic pull) results in the construction of a DsDdPullProcessor object and the invocation of its ServiceRequest operation. Concurrently with this, DsSrWorkingCollection invokes the WaitForCompletion service, which is possible because of the asynchronous return of the DsDdDistRequest via the Distribute service, and may be desired if the initiator (DsSrWorkingCollection in this instance) wishes to know when the request completes.

In processing the request TransferFiles is next invoked, which results in invocation of Allocate to obtain space in the user pull area. Additionally, CreatePackingSlip is called to create the list of items distributed with associated information about the distribution, which is later used by GetPackingListFile to incorporate the information into the EcNotification mail message sent to the user. GetNextItem and DistFrom are called iteratively to respectively get each item to be distributed and transfer it to the user pull area. Upon completion the mail message is sent, the request is removed from the request list, and the WaitForCompletion returns.

Assumptions

- The user desktop has an active dataserver session.
- The job's working collection is populated with the correct ESDTs.
- The distribution interface is up on the client's desktop

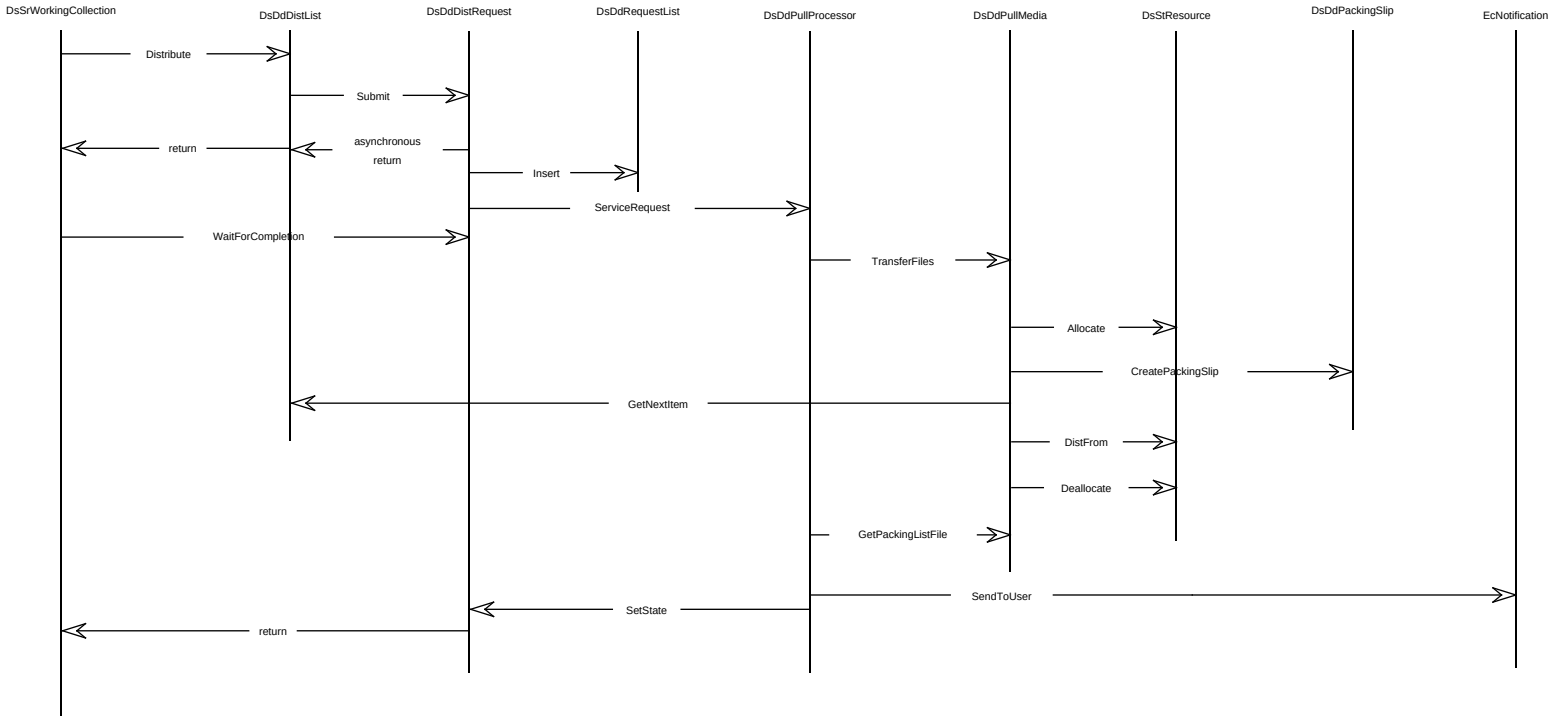


Figure 7.4.2-1. Electronic Pull of Data

7.4.3 Electronic Push of Data

Summary

Figure 7.4.3-1 depicts the distribution of data via electronic push. The scenario has only a few differences from that for Electronic Pull of Data. With Electronic Push the destination resource for DistFrom is the network vice staging disk for Electronic Pull. Also, in Electronic Push GetPackingListFile is not used because the information from CreatePackingSlip is transferred as a file to the push destination vice being included in the mail message to the requestor.

Assumptions

- The user desktop has an active dataserver session.
- The job's working collection is populated with the correct ESDTs.
- The distribution interface is up on the client's desktop

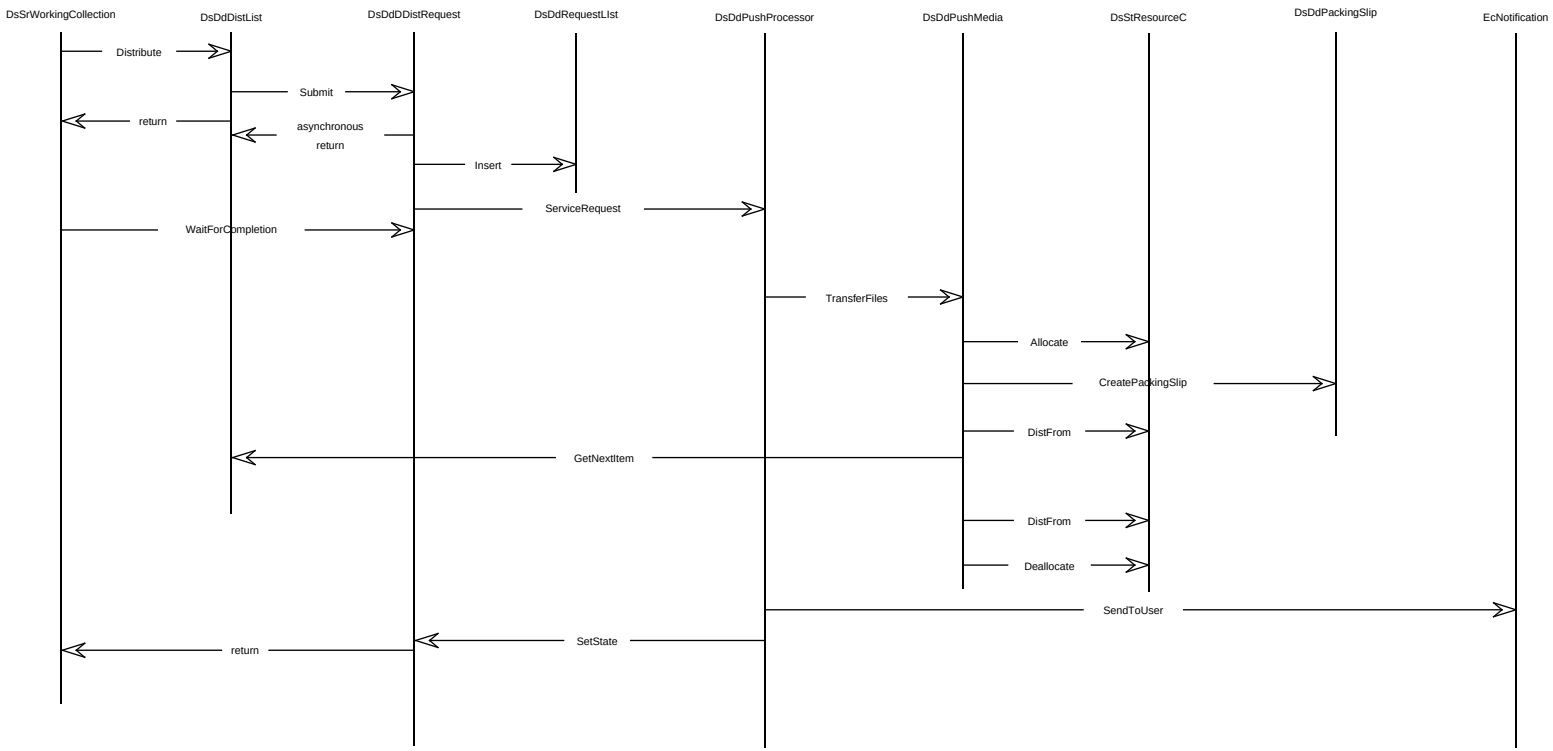


Figure 7.4.3-1. Electronic Push of Data

7.4.4 Physical Distribution to Tape

Summary

Figure 7.4.4-1 depicts the distribution of data via tape. The scenario is the same as that for Electronic Pull up to the Allocate of the resource. Once the resource is allocated, GetResourceID is used to determine the tape drive id, which is used in a Posix system call to write the distribution files in the requested format (tar or cpio) to the tape. The number of volumes written is determined and a packing slip generated, after which the user is notified of completion of the request. Note that for tape (and other physical media) the request is not removed from the request list; this removal occurs only after operations personnel indicate that the distribution product has been shipped to the requester.

Assumptions

- The user desktop has an active dataserver session.
- The job's working collection is populated with the correct ESDTs.
- The distribution interface is up on the client's desktop

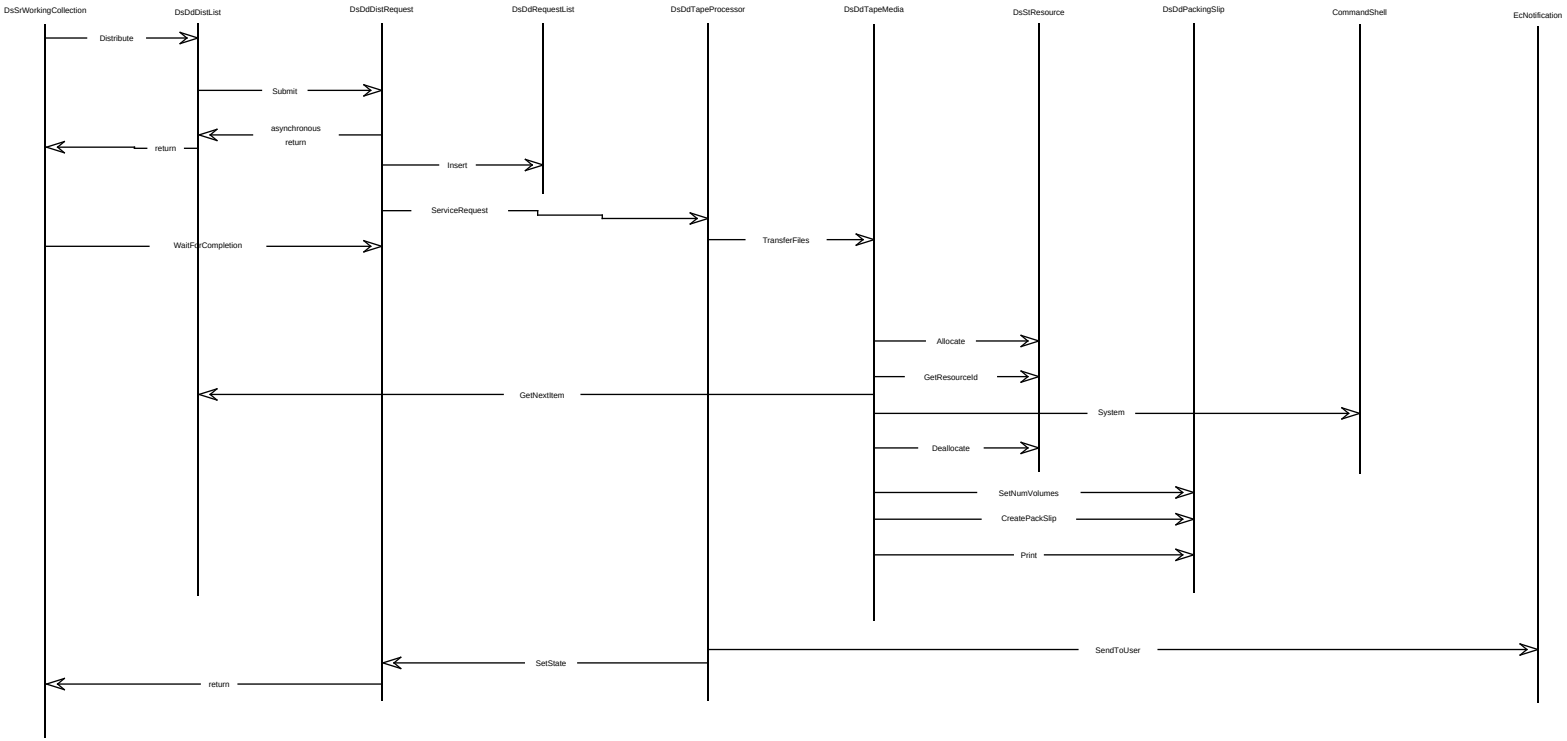


Figure 7.4.4-1. Distribution via Tape

7.4.5 User Abort of a Request Waiting for a Device

Summary

Figure 7.4.5-1 depicts a user-initiated abort of a request which is waiting for a device to become available. The scenario begins with DsSrWorkingCollection invoking the Distribute service, which results in the creation and asynchronous return of a DsDdDistRequest, followed by the submission of that request. The request is added to the DsDdRequestList (a list of all current requests), and the distribution media type (in this example CD) results in the construction of a DsDdCDProcessor object and the invocation of its ServiceRequest operation. TransferFiles is then requested, followed by Allocate, which requests that a CD device be made available. In this example no device is currently available, so the Allocate will block until a device becomes available. While the device availability is pending DsSrWorkingCollection issues an Abort of the request. Because the thread which called Allocate is blocked, the Abort will be serviced by a separate thread, which will request DsStRequestManager to Abort the allocation. When the abort occurs the blocked Allocate thread returns, and SendToUser is called to notify the requestor of the distribution that the abort has occurred. Additionally, the request is removed from DsDdRequestList.

Assumptions

- The user desktop has an active dataserver session.
- The job's working collection is populated with the correct ESDTs.
- The distribution interface is up on the client's desktop

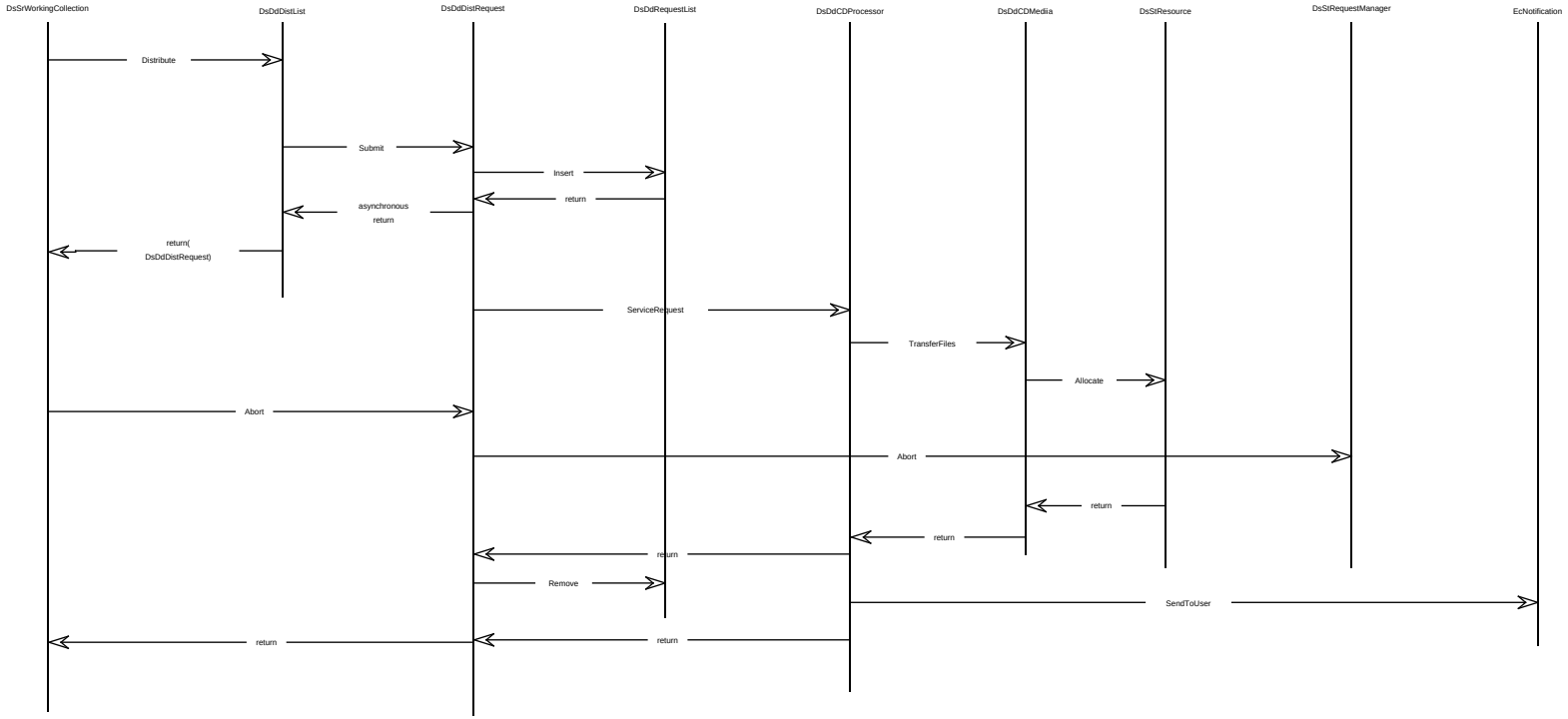


Figure 7.4.5-1. Abort a Request

7.4.6 Tape Fault

Summary

Figure 7.4.6-1 depicts the handling of a recoverable tape fault, such as errors writing to a tape which preclude further use of that tape, but not of the drive. The scenario is the same as Physical Distribution to Tape, except that the tape fault is signaled via an exception, and an exception handler responds to the fault by restarting the operation with a new tape.

Assumptions

- The user desktop has an active dataserver session.
- The job's working collection is populated with the correct ESDTs.
- The distribution interface is up on the client's desktop

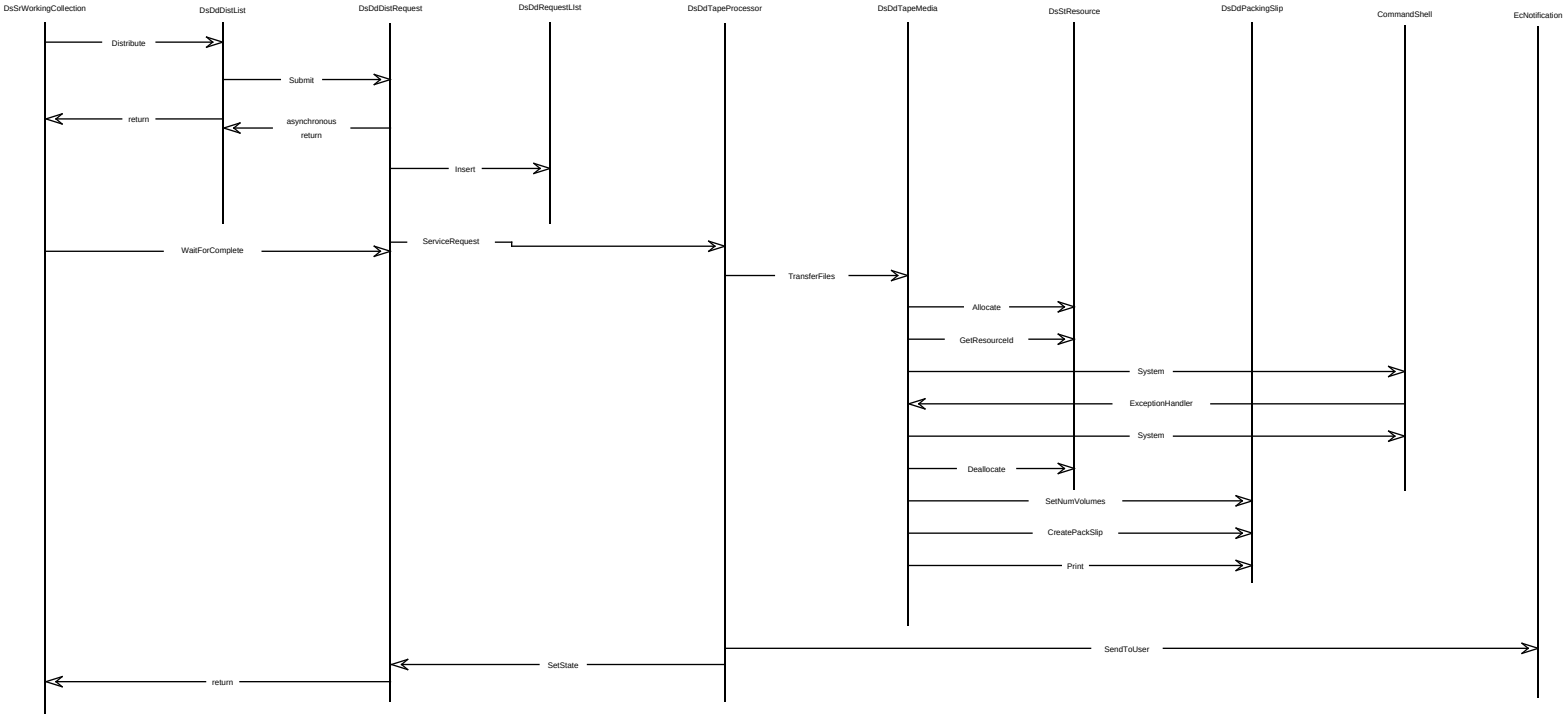


Figure 7.4.6-1. Handling a Tape Fault

7.5 CSCI Structure

Table 7.5-1 shows the components (CSCs) of the CSCI. Each CSC is described and designated as being custom developed code (DEV), off-the-shelf (OTS) or a combination of the two (DEV/ OTS). If the custom developed code will be used for integration purposes, it is identified as WRAPPER.

Table 7.5-1. DDIST's Components

CSC	Description	Implementation
Distribution Products	The classes in this CSC represent the products and outputs generated by the DDIST CSCI.	DEV
Distribution Client Interface	The classes in this CSC are presented to client software for use in initiating, tracking, and manipulating distribution requests.	DEV
Distribution Request Management	The classes in this CSC provide the control and coordination of distribution request processing:	DEV

7.5.1 Distribution Products CSC

Purpose and Description

The classes contained in this CSC represent products generated by the DDIST CSCI. These classes can be further categorized as follows:

- The following classes represent the media via which the data is distributed to the requestor:

  DsDdMedia
  DsDdElectronicMedia
  DsDdPullMedia
  DsDdPushMedia
  DsDdLabeledMedia
  DsDdCDMedia
  DsDdTapeMedia

- The following classes represent products which track and route the data and the media via which it is distributed:

  DsDdPackingSlip
  DsDdShippingLabel

- DsDdRequestList is a software entity which represents the requests presented to the operator so the operator can manage distribution operations.

Objects instantiated from classes in this CSC reside in the distribution server executable.

Candidate products

Not Applicable

ECS white paper references

Not Applicable

7.5.2 Distribution Client Interface CSC

Purpose and Description

This CSC contains classes which are presented to client software for use in initiating, tracking, and manipulating distribution requests, further categorized as follows:

- The following classes define the data items which are to be distributed:

  DsDdDistList
  DsDdDataItem
  DsDdDistFile

- The following classes define the client presentations of the distribution request, with distinct classes for user and operator presentations:

  DsDdDistRequestC
  DsDdOpsRequestC

- DsDdRequestManagerC is the client presentation of the distributed object which implements the OODCE factory model for the creation of distribution requests which are themselves distributed.

Objects instantiated from DsDdDistRequestC, DsDdOpsRequestC, and DsDdRequestManagerC reside in the client process. Objects instantiated from DsDdDistList and DsDdDistFile (DsDdDataItem is an abstract class) may reside in either the client process or the distribution server process.

Candidate products

Not Applicable

ECS white paper references

Not Applicable

7.5.3 Distribution Request Management CSC

Purpose and Description

The classes in this CSC provide the control and coordination of distribution request processing:

DsDdRequestManager
DsDdRequestManagerS
DsDdDistRequest
DsDdPrivRequest
DsDdDistRequestS
DsDdRequestProcessor
DsDdCDProcessor
DsDdTapeProcessor
DsDdPushProcessor
DsDdPullProcessor

Instantiations of these objects reside in the distribution server process.

Candidate products

Not Applicable

ECS white paper references

Not Applicable

7.6 CSCI Management and Operation

The materials in the following paragraphs discuss the management and operations of software components discussed in section 4.5.

7.6.1 System Management Strategy

The DDIST CSCI is designed to provide robust data distribution services to external data providers and requesters. Specifically, the design goal of the DDIST CSCI is to always return status (successful or unsuccessful) for every received request. To accomplish that goal, the CSCI follows ECS project guidelines for:

- Process startup and shutdown;
- Error detection and reporting;
- Fault tolerance and error recovery

7.6.1.1 Startup/Shutdown

MSS provides life-cycle services for system startup and shutdown. The DistributionServer process acts as an "object factory". As such the process instantiates objects in process threads (pthreads) when a request is serviced. The DistributionServer process is started as a standalone process at system startup. The primary role of the DistributionServer Process is to provide a distribution mechanism for physical media and electronic requests.

7.6.1.2 Error Detection and Reporting

The Data Distribution CSCI is designed for primarily automated operations with little need for operations involvement short of tuning and critical error conditions. CSS and MSS jointly provide event logging services for logging and reporting errors and faults, for browsing error/status logs, and for detecting and reporting critical errors. The Data Distribution CSCI will fully use those services during operations. Errors/status may be reported in two error logs. MSS maintains the first log, the MSS event log. It contains errors/status of interest to operations staff to evaluate system status and to perform trend analysis. The Data Distribution CSCI maintains the second log. The Data Distribution event log contains selected errors/status from the MSS event log (for context) plus highly-detailed debug events. Software maintenance personnel use the Data Distribution event log to diagnose system and software problems in response to trouble tickets.

Non-critical errors encountered during processing that will be handled at the application level will be fully resolved and enumerated during development. Major conditions that require operator intervention and/or are considered catastrophic in the processing of requests are listed in Table 7.6.1.2-1.

Table 7.6.1.2-1. DDIST Error Categories

Error Category	Actions to Be Taken
Initialization File/ Environment Corrupt	This would be seen during a system startup process and would result in one or more executables not starting. Operations staff evaluate the condition and correct.
DB Connection Dropped	This could be a serious failure of the Data Base or a short lived problem with the connection. Operations (DBA) would need to evaluate the problem, possibly restarting the Data Base and the Data Server processes.
Internal Queue Overflow	Errors reported along these lines represent a very poorly tuned and/or faulty system. This type error would represent potential loss of service requests. Operations staff would immediately throttle back system processing thresholds and write a trouble ticket for future off line analysis and tuning.
Unable To Allocate Disk Space	Unable to allocate working storage space using Data Server STMGT CSCI services. This is another system of a poorly tuned system. Report alert to operations staff who would immediately lower system thresholds for requests. Operations staff would analyze system off-line and tune.
Unable To Allocate Distribution Device	Unable to allocate a media distribution device due to device off-line, device powered-off, or device in use by another process. This could indicate an operations problem, or a maintenance problem. Report alert to operations staff who would check device status, perform corrective action or notify maintenance.
Communications Link Dropped	This could be a serious failure of an internal component, CSCI, or an external component. Operations would need to analyze a local problem to determine in a hardware component or local CSCI has failed. Operations would coordinate external problems with the appropriate site.

7.6.1.3 Fault Tolerance and Error Recovery

Once a service request is accepted from a client (client being defined as any service requester, not just the ECS Desktop Client), it is the Data Distribution CSCI's design goal to complete the request processing and return status (successful or unsuccessful) to the requester. The Data Distribution CSCI is built on the model of checkpointing processing at the command level (within a request) along with the user's distribution item list (context in a manner of speaking). During restart or recovery operations the CSCI will restore a user's distribution item list and resume processing at the next unprocessed command. Therefore, upon establishment of a user session, the DDIST CSCI will have the user's distribution item list checkpointed to the COTS data base. Likewise, Requests (containing Commands) are also checkpointed. After each command is completed, the distribution item list will be checkpointed if it has changed in value or state. After a process or system failure, the checkpointed distribution item list and Requests are automatically restored to the last checkpointed state and processing continues.

Failure scenarios with recovery methods:

a. Failure of the Data Distribution Executable. This process is immediately restarted as a Unix standalone process. The User Distribution Item List and Requests are restored from their checkpointed states. Data Base integrity is verified and transactions rolled back via COTS procedures.

b. Loss of the data base tables used for checkpointing. The data base management system automatically logs transactions to allow restorations of table information. This feature, coupled with DBA generated Data Base backups, provide for recovery. Since high reliability is required in this area, the Data Base tables will be stored on RAID.

c. Failure of the processor on which the DDIST process is running. In general, the processor automatically restarts. Restart of individual processes is handled as a combination of one or more of the above process restarts. If the processor is disabled, the disablement is detected my MSS SNMP services and a backup processor is restarted. The backup processor has full access to the data base tables used for checkpointing. Again, restart of individual processes is handled as described above.

d. Failure of an external application. After a given number of retries to transmit data or via DCE services that alert us of failure of the recipient process, operations staff are notified by means of an alert message. The DAAC operations staff will coordinate to diagnose the failure.

7.6.2 Operator Interfaces

DAAC operations personnel are provided with an X-Windows/Motif-based GUI to access operations data bases and system configuration information. Table 7.6.2-1 highlights the critical DDIST GUI screens.

Table 7.6.2-1. Data Distribution Management GUI

GUI	Description	Data	Operations
Data Distribution Management	Root screen for the following GUIs.	References to: Distribution Operator Control and Distribution Shipping Management GUIs.	Logon Realizes and/or makes the GUIs visible.
Distribution Operator Control GUI	Allows operators to see and manage distribution events	Distribution state events: electronic transmission problems, media problems, disk management problems	View Abort Change Priority Suspend Resume Reprint Packing List

7.6.3 Standard DDIST Reports

Table 7.6.3-1. Standard Distribution Reports

Report Type	Report Description	Intended Audience
Distribution Profile	This report will provide the total number of orders received for the reporting period which is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date.	System Operator Sr. Science Coordinator Resource Planner Performance Analyst Operations Supervisor DAAC Manager
Media Distribution	Distribution of data by distribution media type. For each media type the reported data will include: the distribution of the requested data in Gigabytes, total number of product instances and the total numbers of physical media.	System Operator Sr. Science Coordinator Resource Planner Performance Analyst Operations Supervisor DAAC Manager
Electronic Distribution	Distribution of data via network resources. For each distribution location the reported data will include: the distribution of the requested data in Gigabytes, total number of product instances and the total numbers of requests.	System Operator Sr. Science Coordinator Resource Planner Performance Analyst Operations Supervisor DAAC Manager
Push Utilization	Total volume of data in Gigabytes and the number of product instances pushed.	System Operator Sr. Science Coordinator Resource Planner Performance Analyst Operations Supervisor DAAC Manager
Distribution Error	Is a summary report of the frequency of errors encountered during distribution processing. The reporting period is either the last 24 hours, the previous 7 days, previous 30 days, or number since a given date. Information for each unsuccessful request includes: date/time stamp, request identifier, error code, error message, destination code.	System Operator Performance Analyst Operations Supervisor DAAC Manager

This page intentionally left blank.

8. ACMHW - Access Control and Management HWCI

The Access hardware allows for client access (both the client subsystem and direct “push/pull” user access) to the Data Server subsystem, provides tools and capabilities for system administration, and supports many of the infrastructure requirements of the Data Server. This hardware configuration item controls logical data server access, maintains client sessions, and directs service requests to other appropriate Data Server subsystem configuration items. The Access Control and Management hardware is broken down into two components; Administration Stations (AS) and Access/Process Coordinators (APCs). The number, type, and configuration of the APCs and Administration Stations vary according to site needs and number of data servers supported.

8.1 HW Design Drivers

The design of the Access Control and Management hardware is driven by the heterogeneity of the client server interfaces as well as a large number of sessions and processes in progress at any one time. The component's configuration must be flexible and its hardware expandable in order to meet the requirements. Partitioning of the Access Control and Management hardware in two categories of AS and APC provides modularity to facilitate the expandability.

The management responsibilities performed by AS component are twofold. The first type of responsibility of the AS is to administer the logical data servers, providing logical resource scheduling, arbitration and access control. The second task of the AS is to manage the physical aspects of the subsystem functioning.

Direct user controlled access to data (i.e., remote mounts and data transfer functions) are isolated to the APC component of this CI. This is done to more easily support security concerns of the system, allow for better control and monitoring of user accesses, and to insure that overall system performance and function is not adversely affected by the electronic ingest and distribution functions.

Data that is to be distributed from the data server may need to be buffered for a period of time. This is true when the data is being pre-processed/formatted for distribution, when the data is being pushed out electronically from the system to the user, or whenever the user is pulling/accessing the data from the data server. The buffering of data is done within the locally attached disk Storage. The management of the user access to the data is the function of the APC. Only less critical data is ingested via APCs. Data that is time-critical or requires very reliable and highly available ingest is not ingested here, but enters the DAAC at the Ingest Subsystem.

8.1.1 Key Trade-off Studies and Prototypes

User Modeling data was used as follows.

a. “User Pull Analysis Notebook”, 160-TP-004-001, Dec. 94 - Jan. 95 Question 20 (Dec. 1, 1994), defined the rate (number per minute) of user requests for system services as a function of time of day (across all DAACs).

b. “User Pull Analysis Notebook”, 160-TP-004-001, Dec. 94 - Jan. 95 Question 47 (Jan. 18, 1995), defined the service requests to each layer of the data pyramid by DAAC.

8.1.2 Sizing and Performance Analysis

In Release A AS will run X-terminal applications for APC, therefore a small workstation will be sufficient in Release A with a potential upgrade to higher capacity in subsequent releases. Workstation, rather than X-terminal would allow for greater operational flexibility.

The APC must be sufficiently scalable to accommodate the throughput rates of the electronic portion of the distribution data (as opposed to the hard media distribution). These rates are going to grow substantially with every Release as the data holdings within the Data Server grow. A server class host is appropriate for the anticipated data throughput rates.

8.1.3 Scalability, Evolvability, and Migration to Release B

Future scalability of the Access Control and Management component will be provided by adding comparable capacity workstations or replacing/augmenting them by more powerful workstations as necessary in the future releases. The role of this component is to assure de-coupling of the Data Server management and service provider functions from an underlying physical data Server implementation, such as COTS DBMS, FSMS, hardware structure etc. This de-coupling allows replacement or addition of physical hardware supporting the component to increase performance without any change in functionality.

The scalability required in the APCs will be supplied by adding I/O capability as necessary. No functional changes would be required for such expansion.

When user demand increases in the future for direct data access capabilities, the access space dedicated to users may need to become a virtualized disk (i.e., the disk will be supported by a deeper archive with data movement controlled via an FSMS or similar product) accessible to the user community. Access rights, longevity of data in the pull area, and data purging rules will be driven by site policies and available site resources.

Some of the electronic user pull sessions from APC may be interrupted by the user after placing a data order. The data will be staged for user pick-up at an unspecified later time. Such transaction pattern could lead to the accumulated staged data needs exceeding the capacity of the staging storage. Therefore, where warranted by data traffic patterns in the future, Hierarchical Storage Management (HSM) may be used to establish data de-staging.

8.2 HWCI Structure

The block diagram Figure 8.2-1 illustrates partitioning of the Access Control and Management hardware into two functional modules.

8.2.1 HWCI Connectivity

All external and internal interfaces will be as follows: data interface of the APC to the File Server computer will be *network* type, all control interfaces will be of *network* type. Data flows will occur between the APC hosts and the disk via a channel type interfaces, between the APCs and external recipients via high capacity distribution networks (e.g., ATM), and between AS and assorted Data Server interfaces via moderate capacity networks (e.g. FDDI).

The Data Server subsystem network connectivity is illustrated in Figure 8.2.1-1.

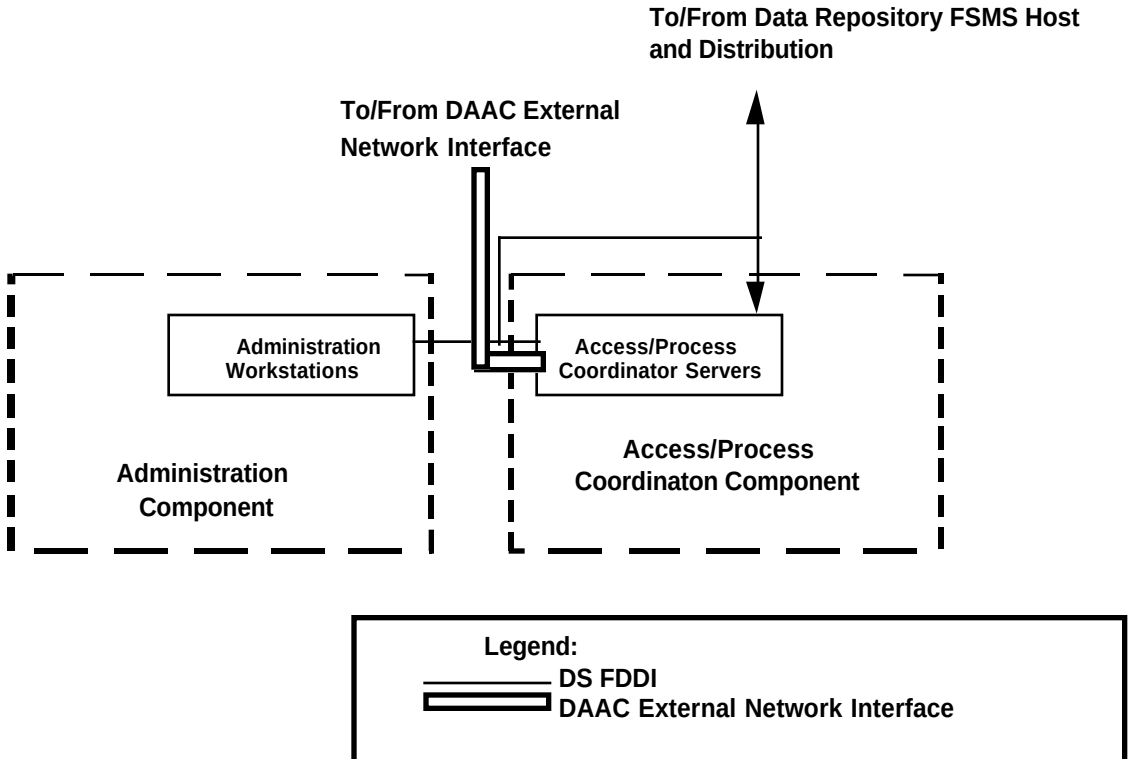


Figure 8.2-1. Access Control and Management HWCI Block Diagram

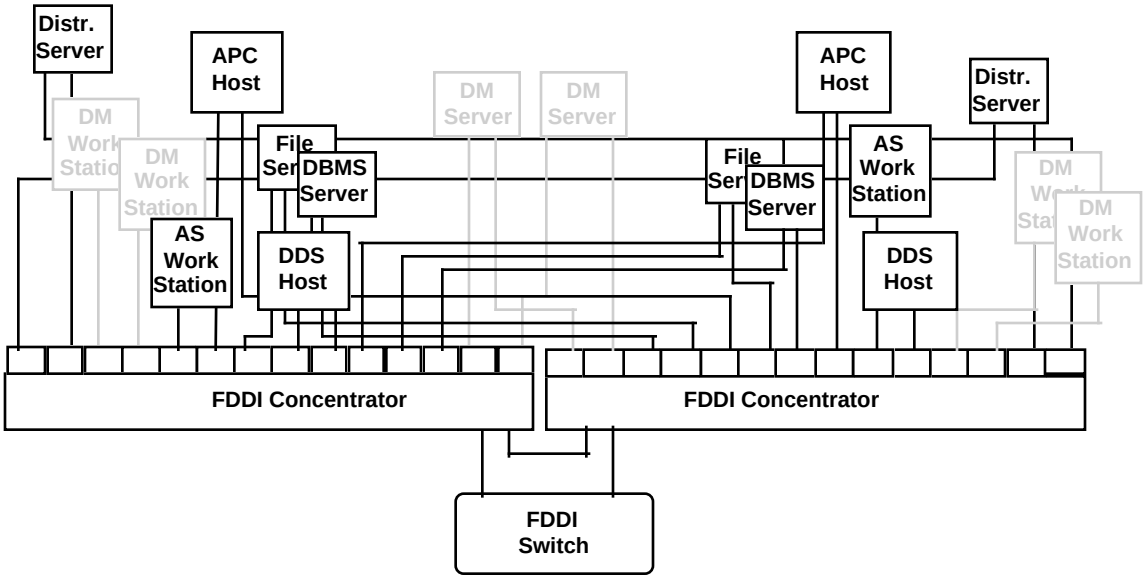


Figure 8.2.1-1. Data Server Network Connectivity

The Data Server servers and workstations will be directly connected to the DAAC FDDI network. The Data Server hosts will be connected to the same FDDI ring as the Data Manager hosts, as is illustrated in Figure 8.2.1-1.

The Data Server processors/servers will contain dual-attached station (DAS) cards, which will be dual-homed to separate FDDI concentrators. This provides redundancy so that full connectivity will exist to the servers even in the event of a concentrator failure. The workstations will contain single-attached station (SAS) cards and each will be connected to a single concentrator, but they will also be split across concentrators so that they are not all connected to the same unit. The FDDI concentrators are in turn connected to the FDDI switch. (Refer to section 5.2 of DID305 Overview Volume 305-CD-004-001 for a general description of DAAC networks.)

8.2.2 HWCI Components

Access Control and Management Hardware Configuration Item consists of two components described below: Administration component and Access/Process Coordination component.

Administration Stations (AS) will host and/or allow access to the Administration Services for one or more data servers. These services will provide Data Server Administrators with the capability to modify and monitor the configuration of the data server. The data server configuration will include resource availability, number and location of components, data server schema, advertised services, data types and archiving strategy. In addition, this set of services will include the capability to perform Archive maintenance functions. Depending on the site configurations, a number of AS may be supported. Because of the modest I/O and processing requirements imposed, these stations will take the form of mid-sized workstations running GUI packages.

The Access/Process Coordinators (APCs) will be used to interface the data server services to the clients. The APCs will support Client session establishment and control. Sessions will be managed by the APCs and are the vehicle through which clients (both user programs and other services) access the data server's object services. Sessions will provide for the management (from a client perspective) of a variety of data server service resources (results sets, cached compute-on-demand data objects [not provided in Release A], search contexts, etc.). These functions will impose a modest computational load and a moderate I/O load on the APC processors. Additionally, a much higher I/O load will be imposed by the APCs functioning as data throughput mechanism for electronic data distribution in Release A.

The APCs may in later releases also provide the compute resources and possible portions of the search engines and tools that operate on data retrieved from the data repositories. Algorithms operating on data at a site may execute on an APC or a resource in the Processing Subsystem. The designation of what algorithms and functions execute where in the system is a site configuration policy and may change over time. A general guideline for initial system analysis would be that generic algorithms that perform non-specific operations on data (i.e., simple geographic subsetting) would be hosted on the APCs while more data specific or computationally intensive algorithms would be hosted in the Processing Subsystem. The sizing of the APC platforms will stem primarily from the I/O responsibilities of the APCs in its function as data throughput mechanism for electronic data distribution, and only as a minor consideration computational capabilities.

Table 8.2.2-1. Access Control and Management HWCI Component Descriptions

Component Name	Class/Type	Comments
Administration	OPS Workstation	Moderately sized workstations with private disk. Host and/or allow access to the Administration Services for one or more data servers. Will be scaled up in subsequent releases by adding workstations or upgrading workstation class.
Access/Process Coordination (APC)	APC Data Server	Initially, mid-capacity servers. Sizing is site specific. Interface the data server services to the clients; provide the compute resources and possible portions of the search engines that operate on data retrieved from the data repositories. Will be scaled up by server class upgrade or additional servers.

8.2.3 Failover and Recovery Strategy

Two AS workstations configured identically will supply AS failover/recovery capability via redundant operation. Either workstation can be used independently as an operator workstation. in the event of failure of one of them the other can assume the full role.

The two APC servers in a cold standby configuration will share a common disk pool. One of the servers will be configured as primary, the other as a cold standby secondary. The active server will maintain the current system configuration files on disk. If a failure on a primary APC is reported by a CSS agent running on that APC or the agent is not communicating within a set period of time due to a catastrophic failure, a failover procedure will be initiated by the CSS. During a failover the backup APC server will read the system status files from the shared disk and begin operation as primary.

There are three types of network failures that may affect the Data Server subsystem. If the FDDI cable between a host and the FDDI concentrator is severed or damaged, then a new cable would need to be installed. No other configuration would be required. If an individual port on the FDDI concentrator fails, then the attached host must be moved to another port, again with no other configuration required. Finally, if the entire concentrator fails, then it will have to be replaced, which can be done rapidly since the units require very little configuration.

Note that the above failures result in service interruption only to the workstations. Since all servers/processors are attached to two hubs, they will communicate as normal in the event of a cable or concentrator fault, and the applications will be unaware of and unaffected by the event.

This page intentionally left blank.

9. WKSHW - Working Storage HWCI

Working Storage (WS) hardware configuration item of the data server supplies a pool of storage used for temporary file and buffer storage within the data Server architecture. In Release B and later WS may also be used to support the higher levels of a hierarchical storage scheme that utilize other data repositories as lower levels in the storage schema. Any data that resides in WS and is not designated as temporary data will be copied to a permanent data repository (see DRPHW - Data Repository HWCI) and maintained there.

WS provides the staging capacity for data acquires and inserts. Because of its role at the higher levels of the archiving hierarchy, WS may hold production related data that is to be accessed in the near future to increase performance. WS also improves performance by retaining copies of frequently accessed data that has been copied to the deeper archives and servicing data requests for that data in a faster, more efficient manner. In this regard WS behaves like a cache for frequently accessed portions of the deep archives.

9.1 HW Design Drivers

The principal driver behind the WS is its ability to hold the data for the ease of access by the requesting components. Therefore, although the WS portion of the Data Server architecture contains limited amounts of storage as compared to the Permanent Data Repository, the WS allows for faster access and relatively high throughput rates.

In order to assure availability of WS in the light of its moderate capacity intelligent data purging methods will be exercised. Data retrieval profiles will be used to predict aging of the data so that data not likely to be requested in the pre-set time frame will be de-staged to the secondary tier (Release B and later) or removed from the WS altogether.

Working storage serves as a buffer to the Data Processing Subsystem (DPS) for high performance data exchange between the Data Server Subsystem and the DPS. It is also used as a storage buffer for data subsetting (post Release A) and ingest that are performed in the Data Server subsystem (TSDIS ingest, some of the ancillary data in Release A, and V0 data migration).

The logical configuration of the WS component is consistent at all DAAC sites. The physical instantiation of the WS hardware will vary in later Releases for the larger and smaller sites due to the variation in the data storage capacities and data rates supported at the sites.

9.1.1 Key Trade-off Studies and Prototypes

In Release A WS will be used primarily for file server data transfer buffering. The studies listed below explored the concept of shared storage, that, although not used in Release A, may be applicable to the subsequent releases.

"Network Attached Storage. Concepts and Industry Survey for the ECS Project" White Paper, 440-TP-009-001. The study identified candidate methods and technologies of network attached storage for a distributed archive and examined the associated implementations issues and risk factors.

"Network Attached Storage Technologies" Study (Draft). The paper examined some of the design alternatives that can be used to satisfy the SDPS internal data transfer requirements. It examined the options of network attached storage implementation as a method of optimizing overall SDPS design. The focus was on the issues of network attached storage within SDPS. Network attached Redundant Array of Inexpensive Disks (RAID) pools and network attached Automatic Tape Library (ATL) were discussed.

9.1.2 Sizing and Performance Analysis

The capacity of the working storage disk for Release A use was determined based on the Processing Subsystem's predicted needs for data pre-staging and production volumes and patterns, staging needs for the electronic distribution, and the minimum FSMS staging requirements. The Processing Subsystem activity assumptions were based on the Performance Modeling group's dynamic analysis of the AHWGP data for *epoch e*. For sizing of the distribution disk area AHWGP data for *epoch e* was used of the size of the largest data granule for TRMM products.

9.1.3 Scalability, Evolvability, and Migration to Release B

The overall approach to increasing the capacity of the WS is "horizontal" rather than "vertical" in nature. The size of the temporary repository will be increased, if necessary, by the use of additional modular storage units rather than by solely relying on such units becoming denser and higher in capacity and throughput as part of technology evolution.

Segmentation of Repositories - The use of multiple file servers and the mixing of storage solutions like FSMS based systems in the architecture lends itself to easy scaling. Adding more file servers can enhance both file handling bandwidth and processing capabilities. The flat common access nature of the robotic based tape repositories allows added file servers to gain access to the data without burdening existing file servers.

Performance Enhancements and Scaling - The architecture maintains performance levels as increasing number of files and I/O requirements are added by allowing for growth in the various tiers of Working Storage. By enlarging Working Storage a high "cache hit" rate can be maintained in the higher tiers of the architecture, thereby keeping the deep archive accesses down to a minimum. Intelligent purging strategies, briefly discussed in HWCI Design rationale, will also be instrumental in reducing capacity demand on the WS in the future Releases.

9.2 HWCI Structure

The DPS will place processed data in the WS for continued access in further reprocessing and for transfer to the permanent data repository. The data may be retained in the WS for some time for repeated access. After that time the data in the WS is deleted with intelligent purging strategies playing dominant role in establishing deletion mechanisms. "High water mark" could be a secondary back-up consideration in conjunction with intelligent purging record base and an operational schedule. At this time a copy of all the permanent data is already in residence in the Permanent Data Repository. The data may be copied to WS again if requested by the DPS for reprocessing.

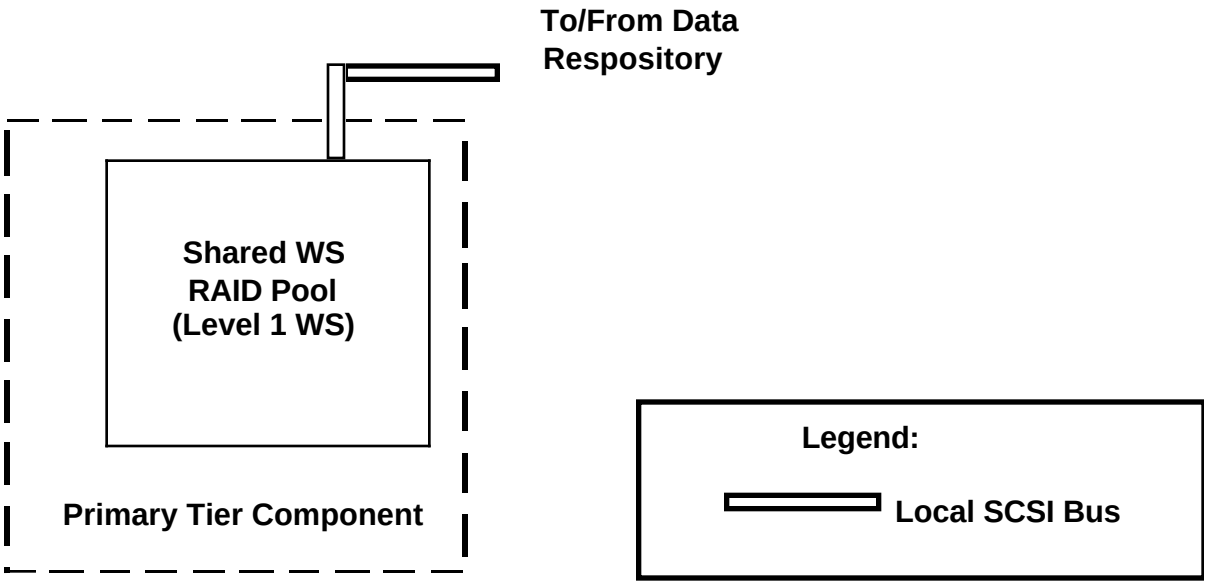


Figure 9.2-1. Working Storage HWCI Block Diagram

9.2.1 HWCI Connectivity

The Data Server servers and workstations will be directly connected to the DAAC FDDI network. The Data Server hosts will be connected to the same FDDI ring as the Data Manager hosts, as is illustrated in Figure 8.2.1-1.

The Data Server processors/servers will contain dual-attached station (DAS) cards, which will be dual-homed to separate FDDI concentrators. This provides redundancy so that full connectivity will exist to the servers even in the event of a concentrator failure. The workstations will contain single-attached station (SAS) cards and each will be connected to a single concentrator, but they will also be split across concentrators so that they are not all connected to the same unit. The FDDI concentrators are in turn connected to the FDDI switch. (Refer to section 5.2 of DID305 Overview Volume 305-CD-004-001 for a general description of DAAC networks.)

9.2.2 HWCI Component Description

For the purposes of sizing and technology choices, product data was assumed to be inserted into the data server and would initially be staged in Working Storage on the staging disks. All product data would then be copied into the deep archive. The copy of data was placed in Working Storage anticipating near term (i.e., 0-3 hours) data acquires that will be required for production processing and the fulfillment of subscription requests.

In subsequent Releases Working Storage may be tiered with the first tier component configured using RAID disks. Potential candidates for the second tier component of Working Storage, all of which would be robotic based and erasable in nature, include magneto-optical disk storage (1 GB platters), smaller tape formats (i.e., serpentine, such as IBM's New Tape Product (NTP), Digital Linear Tape (DLT), smaller capacity helical scan tapes, etc.), and potentially cheaper and slower

magnetic disks. The access patterns required for this data (i.e., random access vs. sequential access and processing) will be a large factor in determining the technologies to be used at a site.

The per site estimates of working storage requirements are shown in the appendices.

Table 9.2.2-1. Working Storage HWCI Component Descriptions

Component Name	Class/Type	Comments
Working Storage Primary Tier	RAID (host attached)	RAID storage amount will grow in the Releases following A. The final RAID capacity at Release D is subject to RAID affordability as well as architectural considerations.

9.2.3 Failover and Recovery Strategy

Fault tolerance of the WS will be provided by using RAID for storage implementation. The RAID will provide degraded mode of operations for a single disk failure.

10. DRPHW - Data Repository HWCI

Data Repositories (DRs) are the hardware components that store and maintain data permanently. Different technologies can be used to instantiate DR depending on the volume and type of data to be stored, the access patterns on this data, and any additional unique requirements that may be imposed on the repository (i.e., data maintenance requirements, backup and restore functions, media management and control, etc.).

DRs will be classified as "permanent", meaning that the services necessary to monitor and maintain data integrity for large data holdings will be supported by this repository's storage technology. As a general configuration rule, every site will have at least one DR in the component designated as a permanent repository. A copy of all data at a site that is not considered temporary will eventually be maintained in a site's permanent DR.

10.1 HWCI Design Rationale

DR hardware of the data server is designed for expandability and evolvability. The expandability is very important in the light of unprecedented data volumes that are received, processed, stored, and distributed by the larger DAACs. The evolvability must be provided because of the historically rapid evolution and aging of storage technologies.

Among other design considerations is the complexity that is introduced by the different requirements for sizing of the DR portion of different DAAC sites. In the subsequent Releases the DR storage hardware will differ at the large and small DAAC. At that time, a single DR may have heterogeneous storage elements within it in order to best accommodate data types at different layers of the data pyramid.

10.1.1 Key Trades and Analysis

"Storage Technology Contingency Plan for the ECS Project" TRADE 193-0111TPP; DRAFT The trade study outlined a plan for an organized approach to dealing with the potential unavailability of the 3480 optical tape cartridge-based large volume archive storage technology originally proposed for use in the ECS Data Server Data Repository. The plan included several implementation scenarios. In each of the proposed scenarios the schedule impact, cost, and reliability issues were examined. The plan was the first step in a progression of analysis leading to the selection of the optimal archive architecture. The scenarios resulted from the storage market survey and initial vendor meetings and evaluations. The overall approach tied the scenario development to the four deliverable releases.

***"Storage Technology Insertion Plan"* White Paper *420-WP-003-001*.** This document reviewed existing secondary storage technology solutions with the purpose of outlining technology use and insertion strategy for the ECS Data Server Permanent Data Repository over Releases of ECS project. The document reflected both past and ongoing analyses of the project requirements and the storage technologies market. One of the primary objectives of the design of the archive portion of the SDPS is to allow for an expandable and evolvable facility.

In the course of past analysis a large number of solutions were found unacceptable because they lacked in scalability, performance, or both. As a result of that analysis and of further refinement of the SDPS architecture, the number of appropriate storage solutions narrowed to just a few logical candidates.

One significant development resulting from the analysis was a move away from a homogeneous archival implementation starting with Release B or later. Some combination of linear and helical scan magnetic tape will be required in order to take advantage of the performance strengths of each of the technologies.

"Network Attached Storage Technologies" Study (Draft). The paper examined some of the design alternatives that can be used to satisfy the SDPS internal data transfer requirements. It examined the options of network attached storage implementation as a method of optimizing overall SDPS design. The focus was on the issues of network attached storage within SDPS. Network attached Redundant Array of Inexpensive Disks (RAID) pools and network attached Automatic Tape Library (ATL) were discussed.

"DBMS Benchmark Report", 430-TP-003-001. Reported the results of the benchmarking study of four DBMS comparing performance of the products relative to each other.

10.1.2 Scalability, Evolvability, and Migration to Release B

Analogously to the methods used in keeping the Working Storage easily scalable, the scaling strategy in the Data Repository is primarily "horizontal". Modularity assures that no functional perturbations occur as a result of scaling.

Increasing Secondary Storage Capacity - Secondary storage capacity within the permanent repository will be increased via two methods: 1) increasing the number of storage units and the corresponding number of robotic arms and tape drives; 2) increasing the capacity of the individual tape cartridges and the throughput rate of the supporting tape drives. The first method is assured through the proper expansion planning through the Releases of the system and archive architecture that allows such addition of hardware. The second method is dependent primarily on further evolution of technology.

The archive, however, is designed in a modular way allowing migration to more efficient technologies in the future with minimal disruption to the normal functioning. See the "Storage Technology Insertion Plan", summarized in 10.1.1. One concrete example of a possible future technology migration would be in the event of market availability of suitable cartridge based optical tape. The deep archives can also scale by the use of heterogeneous storage devices tailored to specific data types and their storage needs. By storing data in hardware/software formats that are optimized to the retrieval patterns seen by that data, overall high performance can be maintained.

Another degree of modularity is introduced by the use of a Volume Server logically separate from the File Server. While at smaller DAAC sites the two may reside on the same physical server host, at larger sites File Server and Volume Server will likely reside on distinct hardware hosts.

Performance Enhancements and Scaling - The architecture maintains performance levels as increasing number of files and I/O requirements are added by allowing for growth in the various tiers of storage as well as Working Storage. By enlarging Working Storage a reasonable "cache hit" rate can be maintained in the higher tiers of the architecture, thereby keeping the deep archive accesses down to a manageable level.

The deep archives can also scale by in the use of heterogeneous storage devices tailored to specific data types and their storage needs. By storing data in hardware/software formats that are optimized to the retrieval patterns seen by that data, overall high performance can be maintained. An example would be allocating larger sized files to a high throughput storage device, such as helical scan tape. Although an initial access penalty is incurred of a helical scan drive head spin up, high transfer rate over the length of a sizable file compensates for the time loss. Conversely, smaller files will either have to be blocked together for storage to realize a throughput benefit, or placed on a storage media that imposes less penalty on access, even if the throughput rate is comparatively lowered.

10.2 HWCI Structure

The sizing of Release A Data Repository components hardware was done using dynamic modeling and based on the assumptions outlined below.

a. AHWGP (baseline 1/95) data was used for the GB/day flow estimates. The data was assumed to cumulate over the time period of a year from the TRMM activation (17 Aug. 1997 through 17 Aug. 1998).

b. V0 PDR Baseline holdings based on quantities for the data sets in "12/1996 (Release A Initial Operations)" plus quantities for the data sets defined in "12/1997 (Release B Initial Operations)".

c. Baseline 1/95 TRMM TSDIS estimates.

d. No media capacity is allowed for data backup at this time.

The FSMS host platform will be selected on the basis of FSMS/platform compatibility, available memory cash and I/O bandwidth.

The Data Base Management System (DBMS) Repository component was sized as follows:

A multi-CPU SMP DBMS Server was selected based on the DBMS software manufacturer's compatibility recommendations, benchmark data, and project benchmarking activities.

The disk was sized to hold the core metadata associated with TRMM data as well that associated with the V0 data sets identified for migration within release A. The products lists have been derived from the DAAC instrument teams representatives. All products are assumed to conform to the Proposed ECS Core Metadata Standard v2.0, 420-TP-001-005, December 1994. The metadata sizing has been calculated from the Metadata Expected with each granule table on page 94 in Core Metadata Standard v2.0

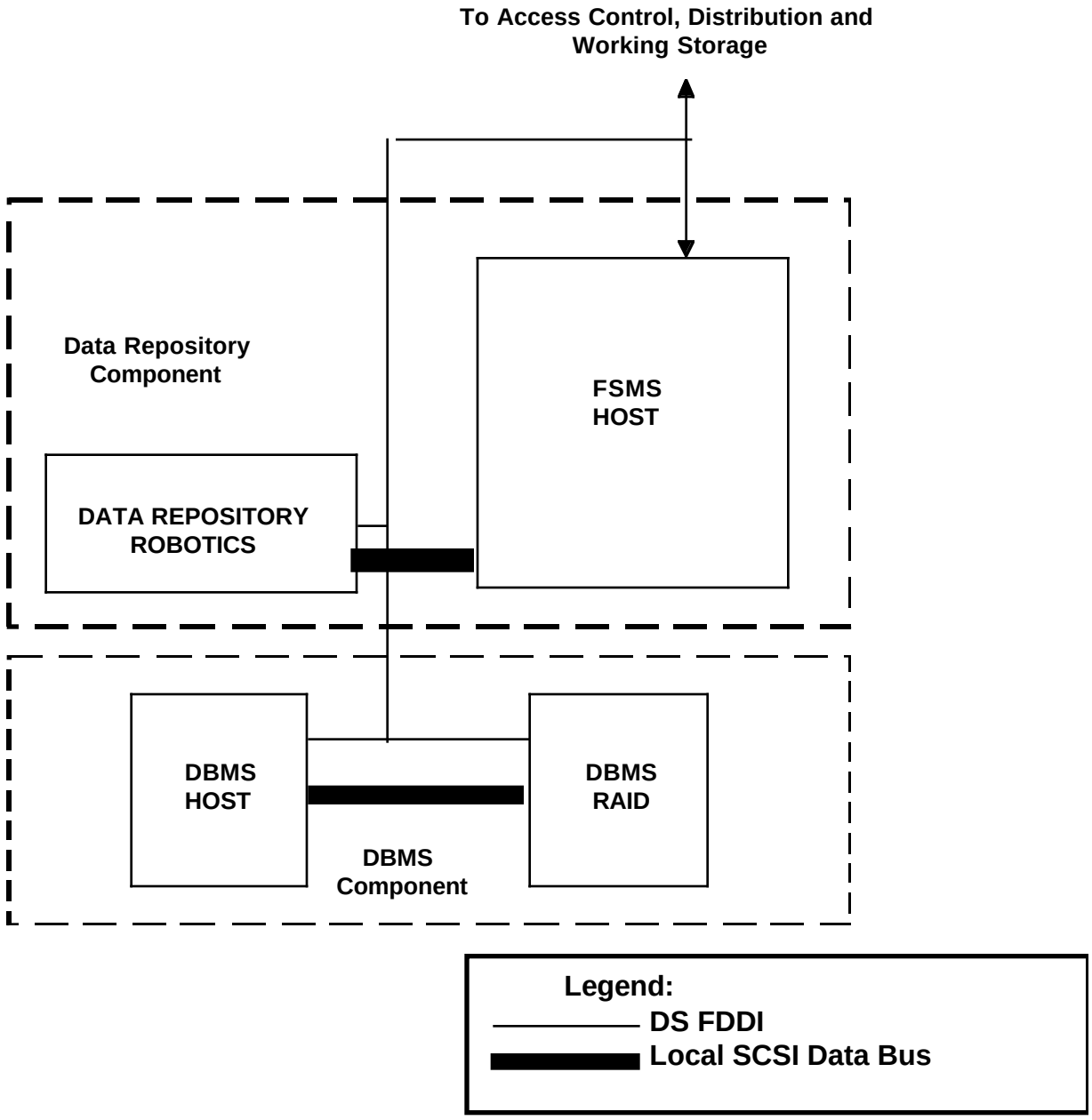


Figure 10.2-1. Data Repository HWCI Block Diagram

10.2.1 HWCI Connectivity

The Data Server servers and workstations will be directly connected to the DAAC FDDI network. The Data Server hosts will be connected to the same FDDI ring as the Data Manager hosts, as is illustrated in Figure 8.2.1-1.

The Data Server processors/servers will contain dual-attached station (DAS) cards, which will be dual-homed to separate FDDI concentrators. This provides redundancy so that full connectivity will exist to the servers even in the event of a concentrator failure. The workstations will contain

single-attached station (SAS) cards and each will be connected to a single concentrator, but they will also be split across concentrators so that they are not all connected to the same unit. The FDDI concentrators are in turn connected to the FDDI switch. (Refer to section 5.2 of DID305 Overview Volume 305-CD-004-001 for a general description of DAAC networks.)

10.2.2 HWCI Component Description

The technologies used for permanent data archiving will be used to instantiate one or more data repositories that are designated as Permanent Archive Management (PAM) types. The Permanent Data Storage Technologies study supplies analysis of the applicability of recording technologies to the storage of data of specific data pyramid levels. For the bulk data holdings that form the lower levels of the data pyramid (i.e., level 1a - level 4), large tape based robotic archives coupled with other robotic based media will be the most cost effective and robust method of permanent data storage. Data stored in this manner will be retrieved sequentially to staging and will be characterized by relatively long seek times and high speed streaming reads.

For several of the largest sites the overwhelming necessity, due to the projected size of the data holdings, will be the use of technology with the highest density of the data storage that is reasonably suitable in Release B and thereafter. Under the suitability criteria are the following factors: sufficient reliability, cost effectiveness, and adequate performance levels. An example of currently available COTS products of that kind is helical scan tape technology. Data that comprises the higher levels of the data pyramid may utilize a different data repository technology for permanent storage, specifically the technology used for operational storage and access. Such technologies can be, as an example, faster access linear tape or, for very fast access, RAID banks.

A large portion of the PAM will be realized in robotic assisted tape based storage schemes. FSMS systems have been used in many field applications to date to control and manage large tape based archives such as that for ECS. ECS will use FSMS products within the PAM, with a few specific requirements and design goals in mind that are reflected in the hardware design. A guiding design approach for releases subsequent to Release A is the separation of command and data paths within the PAM architecture through the use of network attached peripherals. This could result in keeping tape to disk I/O from flowing through the CPUs and limiting it to the network/channel connections that exist between the peripheral devices. Logical separation of control and data paths will be maintained to allow for physical separation.

An additional design goal related to FSMS selection is the separation of volume and file server functions. This will allow for smaller common processors to be used for volume serving, allow for different file management systems to be used within the Archive, and will allow for the sizing and choice of file serving processors to be done with greater independence. At the larger sites File Server and Volume Server will likely reside on distinct hardware hosts.

Table 10.2.2-1. Data Repository HWCI Component Descriptions

Component Name	Class/Type	Comments
Data Repository	Archive Robotics	Single tower robotics for Release A.
	Magnetic Tape Drives	Linear tape is considered for Release A, while higher density tape is planned for the subsequent releases.
	SMP Server Workstation (FSMS Host)	Large Workstation with Multiprocessor Capabilities. Single processor will suffice in Release A, but more processors will be brought in the subsequent Releases.
Data Base Management System (DBMS) Repository	SMP DBMS Server	
	RAID (host attached)	DBMS RAID. The quantities are fairly small in the Release A, but grow substantially in later Releases

10.2.3 Failover and Recovery Strategy

A single archive robotic arm will be provided. In the event of failure a repair will have to be performed.

In Release A a quantity of magnetic drives in the archive will be double the calculated required number. Thus, the redundant drive set will provide the failover capability for the primary set.

The SMP Server workstations serving as FSMS hosts will be configured identically and share a single disk bank to supply failover/recovery capability. One of the workstations will be configured as primary, the other as a cold standby secondary. The active workstation will maintain the current FSMS configuration files on disk. If a failure on a primary workstation is reported by a CSS agent running on that workstation or the agent is not communicating within a set period of time due to a catastrophic failure, a failover procedure will be initiated by the CSS. During a failover the backup workstation will read the system status files from the shared disk and begin operation as primary.

The Similarly, two SMP DBMS host machines in a cold standby configuration will share a common disk pool.

There are three types of network failures that may affect the Data Server subsystem. If the FDDI cable between a host and the FDDI concentrator is severed or damaged, then a new cable would need to be installed. No other configuration would be required. If an individual port on the FDDI concentrator fails, then the attached host must be moved to another port, again with no other configuration required. Finally, if the entire concentrator fails, then it will have to be replaced, which can be done rapidly since the units require very little configuration.

Note that the above failures result in service interruption only to the workstations. Since all servers/processors are attached to two hubs, they will communicate as normal in the event of a cable or concentrator fault, and the applications will be unaware of and unaffected by the event.

11. DIPHW - Distribution and Ingest Peripheral Management HWCI

The hardware of the Distribution and Ingest Peripheral Management supports the hard media distribution methods for data dissemination from the system, as well as hard media ingest of data into the system. Hard media distribution and ingest is done by an assortment of data recording peripherals. The selection of the peripherals is per project requirements. At the larger ingest/ distribution sites, such peripherals may be under robotic control.

Data that is to be distributed from the data server will be buffered for a period of 24 hours. The buffering of that data is the function of the Distribution Storage Management hardware in the data server. The second half of the CI's responsibilities is ingest of the hard media distributed data.

11.1 HWCI Design Rationale

Both the data distribution and the data ingest requirements drive the design of this CI. The Distribution and Ingest Peripheral hardware will supply the hard media intersite and user data distribution as well as scheduling and management of such distribution.

Aside from hard media distribution, this CI's responsibility within the SDPS is hard media ingest. Because of the inherent high reliability and availability requirements for the Ingest Subsystem, it is important for both cost and design considerations that the Ingest Subsystem is kept as small in capacity as possible. Placing the responsibility for that portion of ingest with the Distribution and Ingest Management of the Data Server Subsystem relieves the Ingest Subsystem of the necessity to accommodate the lower priority ingest functions at the site.

The ingest and distribution peripherals were selected on the basis of project requirements. The number of each type of peripheral was chosen with regard for RMA. (At least two each of the peripheral devices that were reported by V0 DAACs to be used with any degree of frequency. The 6250 Tape Drives and CD-ROM were reported to be used infrequently, less than once a month, and do not warrant duplicate quantities at each facility.) The aggregate bandwidth of the peripheral devices exceeds the estimated required bandwidth of media based data distribution in Release A. The required bandwidth was computed based on the assumption of the hard media distribution equal in volume to one time archive repository ingest and a 8 hours per day operation of the hard media preparation. V0 re-ingest is included in these figures.

11.1.1 Key Trades and Analysis

Market surveys of data recording and data storage peripherals. Selection of optimal hardware is based on the market surveys.

Analysis of V0 sites. V0 site analysis allows narrowing down of the market available peripherals to those that are currently in use at the existing ECS sites.

11.1.2 Scalability Strategies

Scalability of the hard media ingest/distribution portion of this HWCI will be assured by adding peripheral devices as necessary. Where warranted by the performance and volume requirements, robotics (for example, stackers and jukeboxes) will be used with the peripheral devices.

11.2 HWCI Structure

Staging disk in the Distribution and Ingest CI of the Data Server subsystem serves as a 24 hour buffering pull. The buffering pull is relied upon to optimize performance of the data retrieval for distribution or data processing/archiving after ingest.

Heterogeneous peripheral devices are used here as required for user community needs as well as the need of other DAACs for data exchange via hard media.

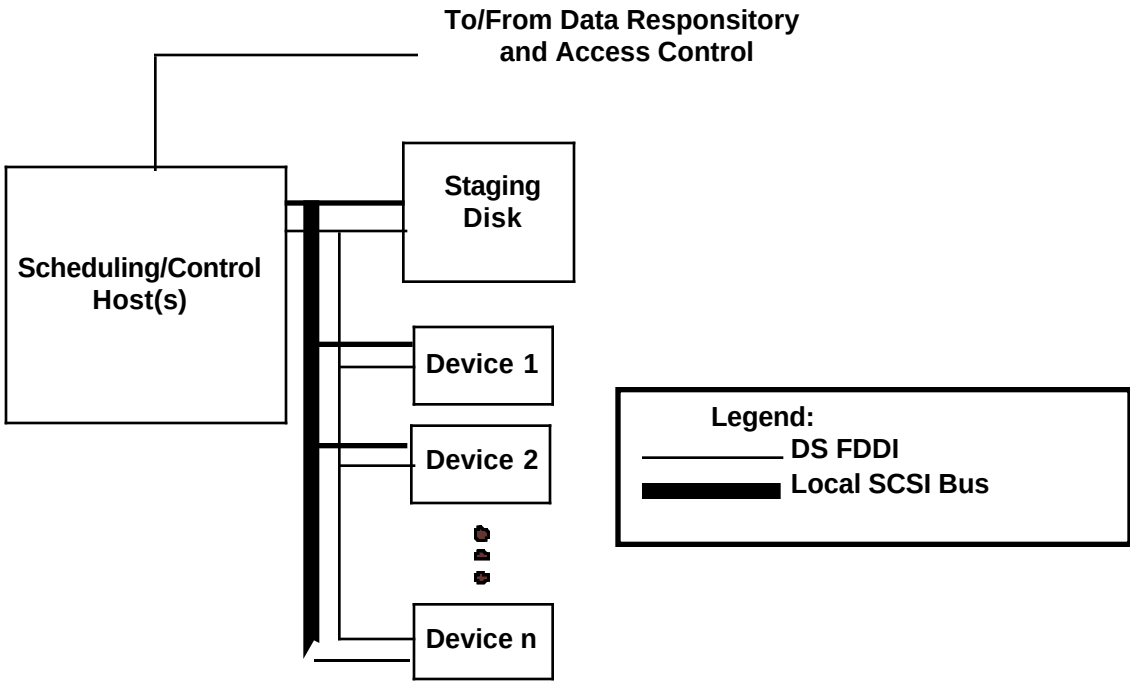


Figure 11.2-1. Distribution and Ingest Peripheral Management HWCI Block Diagram

11.2.1 HWCI Connectivity

The Data Server servers and workstations will be directly connected to the DAAC FDDI network. The Data Server hosts will be connected to the same FDDI ring as the Data Manager hosts, as is illustrated in Figure 8.2.1-1.

The Data Server processors/servers will contain dual-attached station (DAS) cards, which will be dual-homed to separate FDDI concentrators. This provides redundancy so that full connectivity will exist to the servers even in the event of a concentrator failure. The workstations will contain single-attached station (SAS) cards and each will be connected to a single concentrator, but they

will also be split across concentrators so that they are not all connected to the same unit. The FDDI concentrators are in turn connected to the FDDI switch. (Refer to section 5.2 of DID305 Overview Volume 305-CD-004-001 for a general description of DAAC networks.)

11.2.2 HWCI Component Description

The hardware of the Distribution and Ingest Management consists of a variety of recording devices used for both hard media data distribution and hard media data ingest. These devices will be controlled from small workstations. The types of the ingest/distribution media devices will be chosen to accommodate the prevalent types of media in the user community and at other sites. The number of the devices as well the whether or not robotic equipment will be used will be determined for each DAAC site. The use of an HSM with an associated HSM host will likewise be established for each site.

The hardware of this CI constitutes a single component.

Table 11.2.2-1. Distribution and Ingest Peripheral HWCI Component Descriptions

Class/Type	Comments
RAID (host attached)	Hard Media Based Ingest/Distribution Staging Working Storage could be relied upon in Releases B through D for off-loading any peak in ingest/distribution storage need.
6250 Tape Drive	Ingest/Distribution Peripherals: Other peripheral devices may be added as required in Releases beyond A. The actual devices will depend on the user community needs.
8-mm Tape Drives and Stacker	
CD-ROM Jukebox (with Drives)	
Operations Workstation	Support for Data Distribution/Ingest Technician & Mail Clerk

11.2.3 Failover and Recovery Strategy

Workstations will be spared in a cold standby mode. Double quantities of tape and CD drives will be supplied for failover purposes.

There are three types of network failures that may affect the Data Server subsystem. If the FDDI cable between a host and the FDDI concentrator is severed or damaged, then a new cable would need to be installed. No other configuration would be required. If an individual port on the FDDI concentrator fails, then the attached host must be moved to another port, again with no other configuration required. Finally, if the entire concentrator fails, then it will have to be replaced, which can be done rapidly since the units require very little configuration.

Note that the above failures result in service interruption only to the workstations. Since all servers/processors are attached to two hubs, they will communicate as normal in the event of a cable or concentrator fault, and the applications will be unaware of and unaffected by the event.

This page intentionally left blank.

12. DDSRVHW - Document Data Server HWCI

The Document Data Server (DDSRV) provides storage and retrieval services on ESDT related documents and their metadata. Full text and keyword searching is provided, as well as the support for hypertext presentation of document metadata. Document Data Server supports user access to the Guide and Reference Papers.

12.1 HW Design Drivers

The design of the Document Data Server is driven by its function as user interface to the ECS data product collections associated with the TRMM mission.

12.1.1 Key Trade-off Studies and Prototypes

"Portable File Formats" TRADE The trade study analyses major implementation alternatives for standard formats for data other than product earth science data. The essential requirement is portability of such data between platforms. Several document data formats will most likely be adopted.

12.1.2 Scalability, Evolvability, and Migration to Release B

Future scalability of the Document Data Server component will be provided by adding comparable capacity workstations or replacing/augmenting them by more powerful workstations as necessary in the future releases. Disk storage will be increased as necessary.

12.2 HWCI Structure

The block diagram below illustrates partitioning of the Access Control and Management hardware into two functional modules.

12.2.1 HWCI Connectivity

The Data Server servers and workstations will be directly connected to the DAAC FDDI network. The Data Server hosts will be connected to the same FDDI ring as the Data Manager hosts, as is illustrated in Figure 7.6.2.1-1.

The Data Server processors/servers will contain dual-attached station (DAS) cards, which will be dual-homed to separate FDDI concentrators. This provides redundancy so that full connectivity will exist to the servers even in the event of a concentrator failure. The workstations will contain single-attached station (SAS) cards and each will be connected to a single concentrator, but they will also be split across concentrators so that they are not all connected to the same unit. The FDDI concentrators are in turn connected to the FDDI switch. (Refer to section 5.2 of DID305 Overview Volume 305-CD-004-001 for a general description of DAAC networks.)

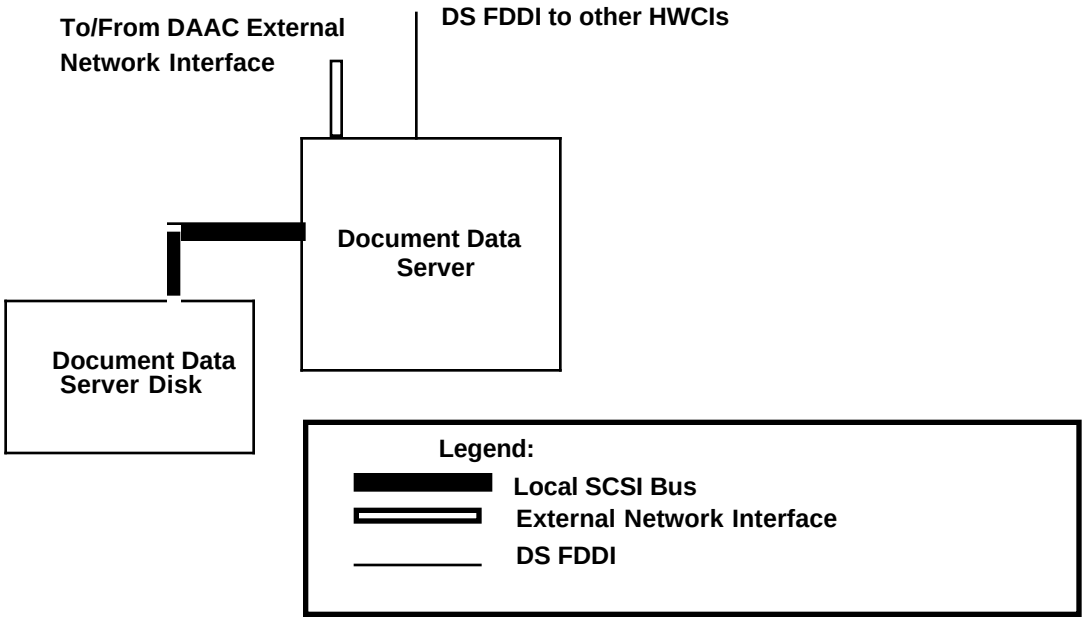


Figure 12.2.1-1. Access Control and Management HWCI Block Diagram

12.2.2 HWCI Components

A 2 CPU SMP server was selected based upon operational experience with the EDF EDHS. A WAIS-like, full text indexer, an http server, and additional custom developed software will reside on this host. The disk was sized to hold the document metadata for the data product collections associated with the TRMM mission, and for the V0 data sets identified for migration. Sizing for document metadata was based on available V0 guide document sizing, and the 2.0 Core metadata baseline. Growth was based on the phased migration of V0 data sets and the TRMM data product collections acquired during Release A operations.

Table 12.2.2-1. Access Control and Management HWCI Component Descriptions

Component Name	Class/Type	Comments
Document Data Server	2 CPU SMP Server	Large Workstation with Multiprocessor Capabilities equipped with two processors. Internal disk (1GB) is sufficient for Release A, but can be supplemented in subsequent Releases.

12.2.3 Failover and Recovery Strategy

Two Document Data server computers will supply redundant functionality. Either machine can perform the function in the event of the loss of the other machine.

There are three types of network failures that may affect the Data Server subsystem. If the FDDI cable between a host and the FDDI concentrator is severed or damaged, then a new cable would need to be installed. No other configuration would be required. If an individual port on the FDDI

concentrator fails, then the attached host must be moved to another port, again with no other configuration required. Finally, if the entire concentrator fails, then it will have to be replaced, which can be done rapidly since the units require very little configuration.

Note that the above failures result in service interruption only to the workstations. Since all servers/processors are attached to two hubs, they will communicate as normal in the event of a cable or concentrator fault, and the applications will be unaware of and unaffected by the event.

This page intentionally left blank.

Appendix. Requirements Trace

The Interim Release 1 (Ir1) and TRMM Development (Release A) Level 4 requirements listed in the following table reflect the state of the RTM database on July 15, 1995.

Table A-1. Requirements Trace (1 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00010	The SDSRV CI shall accept Data Requests for Data that is managed within the STMGT CI.	DsClRequest
S-DSS-00015	The SDSRV CI shall insure that each Data Request includes a User Identifier, a Request Priority, and a Data Identifier.	DsClRequest
S-DSS-00020	The SDSRV CI shall accept Service Requests from clients.	DsClESDTReferenceCollector
S-DSS-00023	The SDSRV CI shall perform services specified by Service Requests.	DsSrSession
S-DSS-00023	The SDSRV CI shall perform services specified by Service Requests.	DsSrSession
S-DSS-00025	The SDSRV CI shall insure that each Service Request includes a User Identifier, a Request Priority, and all other parameters required for that request.	DsClRequest
S-DSS-00030	The SDSRV CI shall provide the capability to queue Service Requests prior to their execution.	DsSrQueuedConnection
S-DSS-00040	The SDSRV CI shall provide operations staff the capability to view queued Service Requests.	DsAdRequestInterface
S-DSS-00050	The SDSRV CI shall process each Service Request on the basis of Priority Information specified in the Service Request.	DsSrQueuedConnection
S-DSS-00051	The SDSRV CI shall verify that each Service Request has valid Priority Information.	DsClRequest
S-DSS-00055	The SDSRV CI shall initiate the processing of Service Requests of equal priority in the order in which they are received.	DsSrQueuedConnection
S-DSS-00060	The SDSRV CI shall acknowledge the receipt of Service Requests from local and remote clients.	DsSrConnection
S-DSS-00065	The SDSRV CI shall accept Service Requests from the Data Processing subsystem and, as a result, provide access to Data for the purpose of standard processing.	DsClRequest
S-DSS-00080	The SDSRV CI shall process Data Insert Requests that request the storage of Data Products and associated Metadata.	DsGeESDT
S-DSS-00090	The SDSRV CI shall validate that each Data Insert Request contains a List of Data Files.	DsGeESDT
S-DSS-00095	The SDSRV CI shall return a Reject Notification if a Service Request fails validation.	DsClRequest

Table A-1. Requirements Trace (2 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00100	The SDSRV CI shall allow operations staff to set a threshold for the number of Service Requests to be queued for processing.	DsAdRequestInterface
S-DSS-00110	The SDSRV CI shall provide operations staff the capability to determine the status of any or all existing Service Requests.	DsAdRequestInterface
S-DSS-00120	The SDSRV CI shall accept Status Requests from clients and, if requested, provide Service Request Status for any specified pending Service Requests, in return.	DsClRequest
S-DSS-00130	The SDSRV CI shall accept Status Requests from clients and, if requested, provide Service Request Status for all pending Service Requests submitted by a specified user, in return.	DsClRequest
S-DSS-00140	The SDSRV CI shall validate that a Status Request specifies either a valid pending Request Identifier or a valid User Identifier.	DsClRequest
S-DSS-00150	The SDSRV CI shall accept and process Insert Metadata Requests to insert Metadata into the Inventory.	DsGeESDT
S-DSS-00160	The SDSRV CI shall accept and process Update Metadata Requests to update Metadata that has been previously stored in the Inventory.	DsMdCatalog
S-DSS-00165	The SDSRV CI shall update the Inventory with the updated Metadata that was received.	DsMdCatalog
S-DSS-00170	The SDSRV CI shall accept and process Search Requests to search the Inventory.	DsClESDTReferenceCollector
S-DSS-00190	The SDSRV CI shall provide the capability for operations staff to delete a queued Data Request.	DsAdRequestInterface
S-DSS-00191	The SDSRV CI shall notify the client whenever operations staff deletes a queued Data Request.	DsSrQueuedConnection
S-DSS-00216	The SDSRV CI shall provide the capability for operations staff to submit Service Requests under that user's User Identifier.	DsClRequest
S-DSS-00220	The SDSRV CI shall provide operations staff the capability to cancel any Service Request.	DsAdRequestInterface
S-DSS-00450	The SDSRV CI shall provide Advertisements that indicate the class of data available from the Data Server.	DsDeService
S-DSS-00460	The SDSRV CI shall provide Advertisements that indicate the services available from the Data Server.	DsDeService
S-DSS-00470	The SDSRV CI shall log all access to data in a Data Access Log.	DsSrConnection
S-DSS-00480	The SDSRV CI shall provide the capability for operations staff to view the Data Access Log.	DsAdLog
S-DSS-00500	The SDSRV CI shall provide the capability for operations staff to sort the Data Access Log by time frame, source of access and data type.	DsAdLog

Table A-1. Requirements Trace (3 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00510	The SDSRV CI shall provide the capability for operations staff to select for viewing from the Data Access Log entries related to data type, source of access, or time frame.	DsAdLog
S-DSS-00520	The SDSRV CI shall return a successful completion status to the provider of data only after all data and associated Metadata has been successfully stored.	DsGeESDT
S-DSS-00530	The SDSRV CI shall provide Data Dictionary Information to the Data Management subsystem.	DsDeDD
S-DSS-00540	The SDSRV CI shall provide Schema Information to the Data Management subsystem.	DsDeDD
S-DSS-00550	The SDSRV CI shall provide the capability for operations staff to view Schema Information.	DsAdDataTypeCollector
S-DSS-00560	The SDSRV CI shall provide the capability for operations staff to create Schema Information.	DsAdDescriptor
S-DSS-00570	The SDSRV CI shall provide the capability for operations staff to update Schema Information.	DsAdDescriptor
S-DSS-00610	The SDSRV CI shall provide the capability for operations staff to delete Schema Information.	DsAdDescriptor
S-DSS-00620	The SDSRV CI shall provide the capability to categorize messages to operations staff into informational, warnings or error categories.	DsAdLog
S-DSS-00630	The SDSRV CI shall notify operations staff of any system error or fault.	DsAdLog
S-DSS-00640	The SDSRV CI shall report to operations staff all errors involving file accesses.	DsAdLog
S-DSS-00650	The SDSRV CI shall expect an acknowledgment for all messages sent to internal components of ECS	DsSrConnection
S-DSS-00660	The SDSRV CI shall acknowledge all messages from internal components of ECS	DsSrConnection
S-DSS-00670	The SDSRV CI shall be capable of receiving data from the PRONG CI.	DsClRequest
S-DSS-00680	The SDSRV CI shall be capable of receiving data from the AITTL CI.	DsClRequest
S-DSS-00690	The SDSRV CI shall be capable of receiving data from the PLANG CI.	DsClRequest
S-DSS-00692	The SDSRV CI shall be capable of receiving data from the DDSRV CI.	DsClRequest
S-DSS-00694	The SDSRV CI shall be capable of receiving data from the STMGT CI.	DsClRequest
S-DSS-00696	The SDSRV CI shall be capable of receiving data from the DDIST CI.	DsClRequest
S-DSS-00700	The SDSRV CI shall be capable of receiving data from FOS.	DsClRequest
S-DSS-00702 *	The SDSRV CI shall be capable of sending management directives to DDSRV CI.	DsAdRequestInterface

Table A-1. Requirements Trace (4 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00704 *	The SDSRV CI shall be capable of sending management directives to DDIST CI.	DsAdRequestInterface
S-DSS-00710	The SDSRV CI shall be accept Data Insert Requests from other Data Servers.	DsClESDTReferenceCollector
S-DSS-00720	The SDSRV CI shall accept Metadata Problem Reports.	DsNsMPR
S-DSS-00734	The SDSRV CI shall provide the capability to store Data Availability Schedules.	DsGeESDT
S-DSS-00820 *	The SDSRV CI shall provide a mechanism to control changes to the Configuration Management Data.	DsCnConfiguration
S-DSS-00850	The SDSRV CI shall provide the capability to control access to Data Server services.	DsSrConnection
S-DSS-00860	The SDSRV CI shall inform a client that a requested service is not accessible if the client attempts to access services outside their access level.	DsSrConnection
S-DSS-00870	The SDSRV CI shall allow Data Access Privileges to be configurable by User Identifier and Data Type for read, write, update, delete, and any combination thereof.	DsAdLog
S-DSS-00880	The SDSRV CI shall use the User Identifier of the user on whose behalf a Service Request is issued as the basis for access control decisions.	DsAdLog
S-DSS-00900	The SDSRV CI shall support the interruption of a data base administrative or maintenance activity and its restart without loss of information.	DsDbEngine
S-DSS-00901	The SDSRV CI shall provide tools for database backup and restore.	DsDbEngine
S-DSS-00902	The SDSRV CI shall provide a database management capability that maintains database integrity during concurrent user interactions.	DsDbEngine
S-DSS-00950	The SDSRV CI shall support the processing of Data Requests subject to access controls of read, write, update and delete, singly or in a any combination, based on data types.	DsSrConnection
S-DSS-00960	The SDSRV CI shall support the processing of Data Requests subject to access controls of read, write, update and delete, singly or in a any combination, based on data ownership.	DsSrConnection
S-DSS-00970	The SDSRV CI shall provide the capabilities to add, delete, or modify ECS Metadata to authorized users only.	DsMdCatalog
S-DSS-01060	The SDSRV CI shall send a Notification to a client that issued a Data Request once the Data Product has been produced or when the STMGT CI has made the Data available.	DsClRequest
S-DSS-01070	The SDSRV CI shall respond to a Data Request with a response that shall contain a status and a pointer to the data.	DsClRequest

Table A-1. Requirements Trace (5 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-01090	The SDSRV CI shall maintain a list of all active Service Requests within the Data Server. The list shall include Request Priorities, Distribution Instructions, and all information necessary to process each request.	DsSrSubmittedRequestVector
S-DSS-01100	The SDSRV CI shall provide the capability for operations staff to view the list of active Service Requests within the Data Server.	DsAdRequestInterface
S-DSS-01120	The SDSRV CI shall provide the capability to manage multiple Service Requests from clients.	DsClRequestVector
S-DSS-01130	The SDSRV CI shall provide the capability to process Service Requests asynchronously.	DsSrConnection
S-DSS-01140	The SDSRV CI shall provide the capability to list and status, Service Requests initiated by a client.	DsClRequestVector
S-DSS-01150	The SDSRV CI shall log all Service Requests entered during a client session.	DsSrConnection
S-DSS-01160 *	The SDSRV CI shall provide periodic, asynchronous status messages to the client during the execution of a Search Request.	DsClQuery
S-DSS-01180	For each Data Request, the SDSRV CI shall log the processing performed, the Data Products produced, any supporting data used and the recipient of the data.	DsGeScienceData
S-DSS-01190	The SDSRV CI shall provide the capability for operations staff to view the resources used and allocated by a client.	DsClSubscriptionCollector
S-DSS-01210	The SDSRV CI shall provide Request Status to a client, concerning pending Service Requests, as specified in Appendix A.	DsClRequest
S-DSS-01400	The SDSRV CI shall log the termination of the processing of a Service Request.	DsSrConnection
S-DSS-01405	The SDSRV CI shall log the termination of client session.	DsSrConnection
S-DSS-01430	The SDSRV CI shall log the initiation of the processing of a Service Request.	DsSrConnection
S-DSS-01460	The SDSRV CI shall accept Subscription Requests that specify an action to be taken and an event to initiate the action.	DsSbSubscriptionInterface
S-DSS-01470	The SDSRV CI shall validate Subscription Requests for receipt of data type events.	DsSbSubscriptionInterface
S-DSS-01472	The SDSRV CI shall validate Subscription Requests for change in core metadata events.	DsSbSubscriptionInterface
S-DSS-01480	The SDSRV CI shall validate Subscription Requests for distribution of data actions.	DsSbSubscriptionInterface
S-DSS-01482	The SDSRV CI shall validate Subscription Requests for send notification actions.	DsSbSubscriptionInterface
S-DSS-01484	The SDSRV CI shall validate Subscription Requests for collection of data for later distribution actions.	DsSbSubscriptionInterface

Table A-1. Requirements Trace (6 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-01488	The SDSRV CI shall validate Subscription Requests for a Data Request action.	DsSbSubscriptionInterface
S-DSS-01490	The SDSRV CI shall process Subscription Requests at the occurrence of the specified event.	DsSbRequestHandler
S-DSS-01500	In the event that more than one Subscription is linked to a single event, the SDSRV CI shall process the actions defined in the Subscriptions on a first-come, first-serve basis.	DsSbRegisteredEvent
S-DSS-01510	The SDSRV CI shall provide the capability to notify users when data has been archived and is available for access.	DsSbSubscription
S-DSS-01525	The SDSRV CI shall accept Subscriptions for Data Availability Schedules from the PLANG CI.	DsClSubscription
S-DSS-01530	The SDSRV CI shall provide the capability for Subscriptions to notify users via email or directly to a program interface.	DsSbSubscription
S-DSS-01550	The SDSRV CI shall provide the capability for a user to request notification of data arrival.	DsSbSubscription
S-DSS-01570	The SDSRV CI shall provide the capability for operations staff to view the stored Subscriptions.	DsClSubscriptionCollector
S-DSS-01600	The SDSRV CI shall provide the capability for operations staff to delete any stored Subscription.	DsClSubscriptionCollector
S-DSS-01610	The SDSRV CI shall provide the capability for a user to delete their own stored Subscription.	DsClSubscriptionCollector
S-DSS-01630	The SDSRV CI shall provide the capability to notify a subscriber of QA changes.	DsGeESDT
S-DSS-01640	The SDSRV CI shall provide the capability to notify a subscriber on individual data granule basis.	DsGeESDT
S-DSS-01760	The SDSRV CI shall log all reported error conditions.	DsSrConnection
S-DSS-01760	The SDSRV CI shall log all reported error conditions.	DsSrConnection
S-DSS-01770	The SDSRV CI shall log all reported warning conditions.	DsSrConnection
S-DSS-01780	The SDSRV CI shall be capable of canceling the execution of a Service Request.	DsClRequest
S-DSS-01800	The LaRC DAAC SDSRV CI shall support TRMM end-to-end testing 9 months before TRMM launch.	SDSRV CI
S-DSS-01810	The MSFC DAAC SDSRV CI shall support TRMM end-to-end testing 9 months before TRMM launch.	SDSRV CI
S-DSS-01820	The SDSRV CI shall support TRMM end to end testing	SDSRV CI
S-DSS-01840	The Science Data Server shall accept and validate Data Requests per hour as derived from Section E.6 of Appendix E.	SDSRV CI
S-DSS-01850	The Science Data Server shall be capable of supporting 200% growth in the number of Data Requests it accepts and validates without architecture or design change.	SDSRV CI

Table A-1. Requirements Trace (7 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-01860	The Science Data Server shall support making stored Data Products available on physical media within 24 hours of receipt of a Media Distribution Request.	SDSRV CI
S-DSS-01870	The Science Data Server shall support distributing product QA data produced at the collocated Data Processing Subsystem within 1 hour from the time it is ready.	SDSRV CI
S-DSS-01880	The Science Data Server shall support making archive data associated with a predefined ECS standard format available to the network in that format within an average of 2 minutes.	SDSRV CI
S-DSS-01890	The Science Data Server shall support making archive data associated with a predefined ECS standard format available to the network in a different format within an average of 5 minutes.	SDSRV CI
S-DSS-01900	The Science Data Server shall be capable of receiving a combined maximum number of Data Requests per hour (across ECS) from the Data Management Subsystem and/or the client Subsystem as derived from Section E.6 of Appendix E.	SDSRV CI
S-DSS-01910	The Science Data Server shall be capable of receiving a combined maximum number of Browse Requests per hour (across ECS) from the Data Management Subsystem and/or the Client Subsystem as derived from Section E.6 of Appendix E.	SDSRV CI
S-DSS-01920	The Science Data Server shall support making pre-computed Browse Data available to a requester in 58 seconds after accepting and validating the request in the number of seconds specified in Appendix E.	SDSRV CI
S-DSS-01930	The Science Data Server and Science Management within the Data Server shall be capable of accepting and storing Data Products derived from Section E.1 of Appendix E from the PRONG CI while supporting standard product retrieval and browse data access load	DSS Subsystem
S-DSS-01940	The Science Data Server and Science Management within the Data Server shall be capable of ingesting product data at a maximum rate (three times the nominal rate derived from Section E.1 of Appendix E bytes per day from the PRONG CI until the backlog is pr	DSS Subsystem
S-DSS-01950	The Science Data Server shall support distributing the number of bytes of data per day derived from Sections E.1 & E.3 of Appendix E to the PRONG CI (in support of production) by accepting and validating the number requests per day from the PRONG CI	SDSRV CI
S-DSS-01960	The Science Management within the Data Server shall support distributing the bytes of data per day derived from Sections E.1 & E.3 of Appendix E to the PRONG CI (in support of production) by retrieving and staging the number of bytes per day for the PRON	DSS Subsystem

Table A-1. Requirements Trace (8 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-02000	The ACMHW CI shall be sized to support the bytes/ second rates derived from Appendix E on the electronic data distribution interfaces. (Supports user push/pull electronic distribution)	ACMHW CI
S-DSS-02010	The ACMHW CI shall be sized to support the number of operations/second derived from Appendix E.	ACMHW CI
S-DSS-02020	The ACMHW CI shall be configured to support the SDPS function of local Data Request Submission's Availability requirement of 0.96000 and Mean Down Time (MDT) requirement of <4 hours.	ACMHW CI
S-DSS-02030	The ACMHW CI shall be configured to support the SDPS function of data order submission across DAACs Availability requirement of 0.96000 and Mean Down Time (MDT) requirement of < 4 hours.	ACMHW CI
S-DSS-02032	The ACMHW CI shall support the RMA Requirements specified for the ECS Program.	ACMHW CI
S-DSS-02900	The SDSRV CI shall provide processing services on ECS Data as listed in Appendix F.	DsGeScienceData
S-DSS-03010	The SDSRV CI shall be capable of receiving Calibration Data.	NsNpCalibration
S-DSS-03020	The SDSRV CI shall be capable of receiving Metadata associated with Calibration Data.	NsNpCalibration
S-DSS-03030	The SDSRV CI shall be capable of receiving Science Software Archive Packages.	DsNsScienceSoftwareArchive-Package
S-DSS-03040	The SDSRV CI shall be capable of receiving Metadata associated with Science Software Archive Packages.	DsNsScienceSoftwareArchive-Package
S-DSS-03110	The SDSRV CI shall be capable of receiving Instrument Calibration Data.	NsNpCalibration
S-DSS-03120	The SDSRV CI shall be capable of receiving Metadata associated with Instrument Calibration Data.	NsNpCalibration
S-DSS-03130	The SDSRV CI shall be capable of receiving Instrument Characterization Data.	DsGeESDT
S-DSS-03150	The SDSRV CI shall be capable of receiving Instrument Historical Data.	DsNsHistoricalData
S-DSS-03160	The SDSRV CI shall be capable of receiving Metadata associated with Instrument Historical Data.	DsNsHistoricalData
S-DSS-03170	The SDSRV CI shall be capable of receiving Inventory Data.	DsMdMetadata
S-DSS-03210	The SDSRV CI shall be capable of receiving Production History.	DsNsProductionHistory
S-DSS-03210	The SDSRV CI shall be capable of receiving Production History.	DsNsProductionHistory
S-DSS-03230	The SDSRV CI shall be capable of receiving Production Plans.	DsNsProdPlans
S-DSS-03250	The SDSRV CI shall be capable of receiving QA Statistics.	DsNsQAStatistics

Table A-1. Requirements Trace (9 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-03260	The SDSRV CI shall be capable of receiving Metadata associated with QA Statistics.	DsNsQAStatistics
S-DSS-03270	The SDSRV CI shall be capable of receiving scientific calibration data.	NsNpCalibration
S-DSS-03280	The SDSRV CI shall be capable of receiving Metadata associated with scientific calibration data.	NsNpCalibration
S-DSS-03310	The SDSRV CI shall be capable of receiving TBD correlative data.	NsNpCorrelative
S-DSS-03320	The SDSRV CI shall be capable of receiving Metadata associated with TBD correlative data.	NsNpCorrelative
S-DSS-03350	The SDSRV CI shall be capable of receiving V0 Migration Data in native format	DsNpVersion0
S-DSS-03360	The SDSRV CI shall be capable of receiving Metadata associated with V0 Migration Data in native format	DsNpVersion0
S-DSS-03370	Upon receipt of data types the SDSRV CI shall perform data type specific checking.	DsGeESDT
S-DSS-03380	Upon receipt of valid data types the SDSRV CI shall pass the data to the STMGT CI.	DsGeESDT
S-DSS-03390	The SDSRV CI shall update the Inventory after the Data it received for insertion into its data holdings have passed the validity checks applicable to the respective data types.	DsGeESDT
S-DSS-03412	The SDSRV CI shall interface with the STMGT CI to provide storage for L0 - L4 Data.	DsGeScienceData
S-DSS-03414	The SDSRV CI shall interface with the STMGT CI to provide storage for Ancillary Data.	DsNpAncillary
S-DSS-03416	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with Ancillary Data.	DsMdCatalog
S-DSS-03420	The SDSRV CI shall interface with the STMGT CI to provide storage for calibration data.	NsNpCalibration
S-DSS-03430	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with calibration data.	DsMdCatalog
S-DSS-03440	The SDSRV CI shall interface with the STMGT CI to provide storage for Science Software Archive Packages.	DsNsScienceSoftwareArchive-Package
S-DSS-03450	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with Science Software Archive Packages.	DsMdCatalog
S-DSS-03480	The SDSRV CI shall interface with the STMGT CI to provide storage for instrument calibration data.	NsNpCalibration
S-DSS-03490	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with instrument calibration data.	NsNpCalibration

Table A-1. Requirements Trace (10 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-03500	The SDSRV CI shall interface with the STMGT CI to provide storage for instrument characterization data.	DsGeESDT
S-DSS-03520	The SDSRV CI shall interface with the STMGT CI to provide storage for instrument history data.	DsNsHistoricalData
S-DSS-03540	The SDSRV CI shall interface with the STMGT CI to provide storage for inventory characteristic data.	DsMdCatalog
S-DSS-03560	The SDSRV CI shall interface with the STMGT CI to provide storage for Orbit/Attitude data.	DsNpOA
S-DSS-03570	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with Orbit/ Attitude data.	DsMdCatalog
S-DSS-03580	The SDSRV CI shall interface with the STMGT CI to provide storage for Production History data.	DsNsProductionHistory
S-DSS-03600	The SDSRV CI shall interface with the STMGT CI to provide storage for production plan data.	DsNsProdPlans
S-DSS-03620	The SDSRV CI shall interface with the STMGT CI to provide storage for QA Statistics.	DsNsQAStatistics
S-DSS-03630	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with QA Statistics.	DsMdCatalog
S-DSS-03640	The SDSRV CI shall interface with the STMGT CI to provide storage for scientific calibration data.	NsNpCalibration
S-DSS-03650	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with scientific calibration data.	DsMdCatalog
S-DSS-03650	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with scientific calibration data.	DsMdCatalog
S-DSS-03680	The SDSRV CI shall interface with the STMGT CI to provide storage for correlative data.	NsNpCorrelative
S-DSS-03690	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with correlative data.	DsMdCatalog
S-DSS-03712 *	The SDSRV CI shall interface with the STMGT CI to provide storage for Research results (articles, algorithms, data sets, software).	DsStArchive
S-DSS-03720	The SDSRV CI shall interface with the STMGT CI to provide storage for V0 migration data.	DsNpVersion0
S-DSS-03730	The SDSRV CI shall interface with the STMGT CI to provide storage for Metadata associated with V0 migration data.	DsMdCatalog
S-DSS-03740	The SDSRV CI shall interface with the STMGT CI to provide storage for validated Inventory data.	DsMdCatalog
S-DSS-03745	The SDSRV CI shall interface with the STMGT CI to provide storage for File Format Descriptions (e.g., HDF Spec.).	DsDeESDTDescriptor

Table A-1. Requirements Trace (11 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-03750	The SDSRV CI shall support Schema Information for each Data Type.	DsDeESDTDescriptor
S-DSS-03760	The SDSRV CI Schema Information shall include for each Data Type the structure of that Data Type.	DsDeScienceParameter
S-DSS-03770	The SDSRV CI Schema Information shall include for each Data Type the services available for that Data Type.	DsDeService
S-DSS-03780	The SDSRV CI Schema Information shall include for each Data Type the Data Type Attributes for that Data Type and the Valid Values associated with each Data Type Attribute.	DsDeValid
S-DSS-03810	The SDSRV CI shall have the ability to cancel the advertising of publicly available services.	DsDeService
S-DSS-03820	Each SDSRV CI Advertisement shall identify the service's interface.	DsDeService
S-DSS-03830	Each SDSRV CI Advertisement shall include Service Descriptions.	DsDeService
S-DSS-03860	The SDSRV CI shall be capable of receiving status from the PRONG CI.	DsGeESDT
S-DSS-03862	The SDSRV CI shall be capable of sending status to the PRONG CI.	DsClRequest
S-DSS-03864	The SDSRV CI shall be capable of receiving status from the PLANG CI.	DsGeESDT
S-DSS-03865	The SDSRV CI shall be capable of receiving scheduling data from the PLANG CI.	DsClRequest
S-DSS-03866	The SDSRV CI shall be capable of sending status to the PLANG CI.	DsClRequest
S-DSS-03868	The SDSRV CI shall be capable of sending status to the WKBCH CI.	DsClRequest
S-DSS-03870	The SDSRV CI shall be capable of receiving status from the INGST CI.	GlStatus
S-DSS-03872	The SDSRV CI shall be capable of sending status to the INGST CI.	DsClRequest
S-DSS-03874 *	The SDSRV CI shall be capable of receiving status from the LIMGR CI.	GlStatus
S-DSS-03876 *	The SDSRV CI shall be capable of sending status to the LIMGR CI.	GlStatus
S-DSS-04035	The SDSRV CI shall supply the Data Products listed in Appendix F to the DDIST CI.	DsGeScienceData
S-DSS-04037	The SDSRV CI shall the Metadata associated with the Data Products listed in Appendix F to the DDIST CI.	DsGeScienceData
S-DSS-04040	The SDSRV CI shall supply calibration data to the DDIST CI.	NsNpCalibration
S-DSS-04050	The SDSRV CI shall supply Metadata associated with calibration data to the DDIST CI.	NsNpCalibration

Table A-1. Requirements Trace (12 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-04060	The SDSRV CI shall supply Science Software Archive Packages to the DDIST CI.	DsNsScienceSoftwareArchive-Package
S-DSS-04070	The SDSRV CI shall supply Metadata associated with Science Software Archive Packages to the DDIST CI.	DsNsScienceSoftwareArchive-Package
S-DSS-04090 *	The SDSRV CI shall supply Metadata associated with FDF repaired orbit data for AM-1 instruments to the DDIST CI.	DsGeESDT
S-DSS-04100	The SDSRV CI shall supply instrument calibration data to the DDIST CI.	NsNpCalibration
S-DSS-04110	The SDSRV CI shall supply Metadata associated with instrument calibration data to the DDIST CI.	NsNpCalibration
S-DSS-04120	The SDSRV CI shall supply instrument characterization data to the DDIST CI.	DsGeESDT
S-DSS-04130	The SDSRV CI shall supply Metadata associated with instrument characterization data to the DDIST CI.	DsGeESDT
S-DSS-04140	The SDSRV CI shall supply instrument historical data to the DDIST CI.	DsNsHistoricalData
S-DSS-04150	The SDSRV CI shall supply Metadata associated with instrument historical data to the DDIST CI.	DsNsHistoricalData
S-DSS-04160	The SDSRV CI shall supply inventory characteristic data to the DDIST CI.	DsMdCatalog
S-DSS-04170	The SDSRV CI shall supply Metadata associated with inventory characteristic data to the DDIST CI.	DsMdCatalog
S-DSS-04200	The SDSRV CI shall supply Production History data to the DDIST CI.	DsNsProductionHistory
S-DSS-04200	The SDSRV CI shall supply Production History data to the DDIST CI.	DsNsProductionHistory
S-DSS-04210	The SDSRV CI shall supply Metadata associated with Production History data to the DDIST CI.	DsNsProductionHistory
S-DSS-04220	The SDSRV CI shall supply production plan data to the DDIST CI.	DsNsProdPlans
S-DSS-04230	The SDSRV CI shall supply Metadata associated with production plan data to the DDIST CI.	DsNsProdPlans
S-DSS-04240	The SDSRV CI shall supply QA Statistics to the DDIST CI.	DsNsQAStatistics
S-DSS-04250	The SDSRV CI shall supply Metadata associated with QA Statistics to the DDIST CI.	DsNsQAStatistics
S-DSS-04260	The SDSRV CI shall supply scientific calibration data to the DDIST CI.	NsNpCalibration
S-DSS-04270	The SDSRV CI shall supply Metadata associated with scientific calibration data to the DDIST CI.	NsNpCalibration
S-DSS-04280	The SDSRV CI shall supply spacecraft historical data to the DDIST CI.	DsNsHistoricalData
S-DSS-04290	The SDSRV CI shall supply Metadata associated with spacecraft historical data to the DDIST CI.	DsNsHistoricalData

Table A-1. Requirements Trace (13 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-04300	The SDSRV CI shall supply correlative data sets to the DDIST CI.	NsNpCorrelative
S-DSS-04310	The SDSRV CI shall supply Metadata associated with correlative data sets to the DDIST CI.	NsNpCorrelative
S-DSS-04360	The SDSRV CI shall include granule-specific information as defined in the SDPS Core Metadata Baseline (194-00269TPW).	DsMdCatalog
S-DSS-04370	The STMGT CI shall have the ability to store product specific Metadata.	DsMdCatalog
S-DSS-04380	The STMGT CI shall store the following Metadata: granule id, date and time of storage, physical storage location, data check status and data format type.	DsMdMetadata
S-DSS-04390	Standard Product related Metadata at the Data Server shall include Metadata associated with static subsetted, subsampled, and summary products.	DsMdMetadata
S-DSS-04400	The STMGT CI shall have the ability to store references to calibration data as Metadata for science data.	DsMdMetadata
S-DSS-04420	The STMGT CI shall have the ability to store references to instrument engineering data as Metadata for science data.	DsMdMetadata
S-DSS-04430	The STMGT CI shall have the ability to store references to Science Software Archive Packages as Metadata for science data.	DsMdMetadata
S-DSS-04440	The STMGT CI shall have the ability to store references to data generation software as Metadata for science data.	DsMdMetadata
S-DSS-04450	The STMGT CI shall have the ability to store references to Production History data as Metadata for science data.	DsMdMetadata
S-DSS-04460	The STMGT CI shall have the ability to store references to data recipients as Metadata for science data.	DsAdLog
S-DSS-04470	The STMGT CI shall have the ability to store references to the data production facility as Metadata for science data.	DsMdMetadata
S-DSS-04475	The STMGT CI shall have the ability to store documents and/or data from the DDSRV CI.	DsStArchive
S-DSS-04476	The DDSRV CI shall provide the ability to store documents and/or data.	DsEsESDT; DsEsTypeID; DsCdCSDT; DsCdTypeID
S-DSS-04480	The STMGT CI shall have the ability to store references to QA Statistics as Metadata for science data.	DsMdMetadata
S-DSS-04490	The STMGT CI shall have the ability to store references to reference documentation as Metadata for science data.	DsMdMetadata

Table A-1. Requirements Trace (14 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-04510	The STMGT CI shall provide the capability to logically group a set of granule ids such that the set can be referenced by a single identifier.	DsMdCatalog
S-DSS-04520	The STMGT CI shall provide the capability to validate metadata before insertion into the Inventory.	DsDeValid
S-DSS-04530	The STMGT CI shall provide the capability to validate updated metadata before insertion into the Inventory.	DsDeValid
S-DSS-04540	The STMGT CI shall reject metadata which fails one or more validations constraints.	DsDeValid
S-DSS-04570	The SDSRV CI shall provide services to add to the existing Inventory	DsMdCatalog
S-DSS-04580	The SDSRV CI shall provide services to delete from the existing Inventory	DsMdCatalog
S-DSS-04590	The SDSRV CI shall provide services to modify the existing Inventory	DsMdCatalog
S-DSS-04600	The SDSRV CI shall update the Metadata for a data item whenever an unexpected loss occurs.	DsGeESDT
S-DSS-04610	The SDSRV CI shall update the Metadata whenever a data item is updated.	DsMdCatalog
S-DSS-04640	The SDSRV CI shall provide services to retrieve Metadata from the Inventory.	DsMdCatalog
S-DSS-04650	The SDSRV CI shall accept Search Requests	DsMdCatalog
S-DSS-04660	The SDSRV CI shall provide Result Sets to the client, in response to Search Requests	DsMdCatalog
S-DSS-04670	The SDSRV CI shall support Inventory searches based on the Core Inventory Metadata.	DsMdCatalog
S-DSS-04680	The SDSRV CI shall support Inventory searches based on the Product Specific Metadata.	DsMdCatalog
S-DSS-04690	The SDSRV CI shall support Inventory searches based on a combination of the Core Inventory Metadata and Product Specific Metadata.	DsMdCatalog
S-DSS-04700	The SDSRV CI shall provide Search Results to requesting agencies.	DsMdCatalog
S-DSS-04710	The SDSRV CI shall respond to a query with a null Result Set, if no products in the Inventory meet the specified criteria.	DsMdCatalog
S-DSS-10010	The guide shall be maintained on-line by the DDSRV CI.	DsEsGuide; DsEsGuideTypeID
S-DSS-10030	The DDSRV CI shall support storage, retrieval and searching of documents in HTML format.	DsCdHTML
S-DSS-10040	The DDSRV CI shall accept Documents from the INGST CI.	DsDoClient; DsCtClient; DsDoRequest; DsCtRequest; DsCtCommand; DsDoCommand; DsCtInsertCommand

Table A-1. Requirements Trace (15 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-10050	The DDSRV CI shall provide documents to requesting agencies.	DsDoClient; DsCtClient; DsDoRequest; DsCtRequest; DsCtCommand; DsDoCommand; DsCtAcquireCommand
S-DSS-10051	The DDSRV CI shall provide the capability to add, delete, or modify individual ECS Metadata entries.	DsEsESDT; DsEsTypeID
S-DSS-10052	The DDSRV CI shall generate an update to metadata reflecting changes in data holdings resulting from a purge operation.	DsEsESDT; DsEsTypeID
S-DSS-10053	The DDSRV CI shall generate an update to metadata reflecting changes in data holdings resulting from an unexpected loss.	DsEsESDT; DsEsTypeID
S-DSS-10054	The DDSRV CI shall generate an update to metadata reflecting changes in data holdings resulting from an intra-site data transfer or some other update.	DsEsESDT; DsEsTypeID
S-DSS-10060	The DDSRV CI shall provide access to the ECS guide (documentation/reference material) and guide services.	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID; DsEsReferencePaper; DsEsReferencePaperTypeID; DsDoClient; DsCtClient; DsDoRequest; DsCtRequest; DsCtCommand; DsDoCommand; DsCtSearchCommand; DsCtAcquireCommand
S-DSS-10070	The DDSRV CI shall store, maintain and provide data management services for ECS guide (documentation/ reference material).	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID; DsEsReferencePaper; DsEsReferencePaperTypeID; DsDoClient; DsCtClient; DsDoRequest; DsCtRequest; DsCtCommand; DsDoCommand; DsCtInsertCommand
S-DSS-10080	The DDSRV CI shall provide the capability to add, delete, or modify groups of ECS Metadata entries.	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID; DsEsReferencePaper; DsEsReferencePaperTypeID
S-DSS-10090	The DDSRV CI shall be capable of receiving documentation of processing algorithms used for EOS and other Earth Science Data Products generated by the ECS	DsEsAlgorithmDescription; DsEsAlgorithmDescriptionTypeID
S-DSS-10100	The DDSRV CI shall be capable of receiving references to results of science data quality assessments of EOS data	DSNsQAStatistics
S-DSS-10110	The DDSRV CI shall be capable of receiving bibliography information of published and unpublished literature (as available) derived from the project	DsEsReferencePaper; DsEsReferencePaperTypeID
S-DSS-10120	The DDSRV CI shall be capable of providing cross references between differing studies of the same data	DsGeESDT
S-DSS-10130	The DDSRV CI shall be capable of receiving other documents relevant to quality assessment of EOS data	DSNsQAStatistics

Table A-1. Requirements Trace (16 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-10140	The DDSRV CI shall provide the capability to receive data describing format and media options available for a given data set.	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID;
S-DSS-10150	The DDSRV CI shall be capable of receiving instrument specifications	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID;
S-DSS-10160	The DDSRV CI shall provide the capability to receive summaries of data sets derived from observation logs	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID;
S-DSS-10170	The DDSRV CI shall receive user supplied documents in HTML & ASCII	DsCdCSDT; DsCdTypeID; DsCdASCII; DsCdHTML
S-DSS-10180	The DDSRV CI shall provide the capability to receive data describing subsetting, subsampling, and transformation options available for a given data set.	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID;
S-DSS-10184	The DDSRV CI shall notify operations staff of any system error or fault.	DsDoServer; DsSvServer
S-DSS-10186	The DDSRV CI shall report to operations staff all errors involving file accesses.	DsDoServer; DsSvServer; DsDoClient; DsCtClient; DsDoRequest; DsCtRequest; DsCtCommand; DsDoCommand; DsCtInsertCommand; DsCtAcquireCommand; DsCdCSDT; DsCdTypeID
S-DSS-10190	The DDSRV CI shall receive Guide Data from Version 0 in HTML & ASCII	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID; DsCdCSDT; DsCdTypeID; DsCdASCII; DsCdHTML
S-DSS-10200	The DDSRV CI shall provide the capability to ingest documentation in ASCII text format.	DsCdCSDT; DsCdTypeID; DsCdASCII
S-DSS-10204	The DDSRV CI shall provide the capability to ingest documentation in HTML format.	DsCdCSDT; DsCdTypeID; DsCdHTML
S-DSS-10209	The DDSRV CI shall provide the capability to ingest documentation in Postscript format.	DsCdCSDT; DsCdTypeID; DsCdPostScript
S-DSS-10209	The DDSRV CI shall provide the capability to ingest documentation in Postscript format.	DsCdCSDT; DsCdTypeID; DsCdPostScript
S-DSS-10210	The DDSRV CI shall receive information that describes spacecraft-housekeeping and Ancillary Data parameters stored in the Science Data Server.	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID
S-DSS-10220	The DDSRV CI shall receive Guide Data from Version 0.	DsEsESDT; DsEsTypeID; DsEsGuide; DsEsGuideTypeID
S-DSS-10241	Upon receipt of all supported document formats and descriptive data, the DDSRV CI shall provide storage for the document and descriptive data.	DsEsESDT; DsEsTypeID; DsCdCSDT; DsCdTypeID
S-DSS-10241	Upon receipt of all supported document formats and descriptive data, the DDSRV CI shall provide storage for the document and descriptive data.	DsEsESDT; DsEsTypeID; DsCdCSDT; DsCdTypeID
S-DSS-10250	Upon receipt and successful storage of all supported document formats and descriptive data, the DDSRV CI shall provide access to the document and/or data.	DsCtSearchCommand; DsCtAcquireCommand

Table A-1. Requirements Trace (17 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-10290	The DDSRV CI shall supply documents to the DDIST CI.	DsCtSearchCommand; DsCt-AcquireCommand
S-DSS-10292 *	The DDSRV CI shall receive management directives from the SDSRV CI.	DsAdRequestInterface
S-DSS-10320	The Document Data Server shall accept and validate the number of Distribution Requests per hour derived from Section E.6 of Appendix E.	DDSRV CI
S-DSS-10330	The Document Data Server shall be capable of supporting 200% growth in the number of Distribution Requests it accepts and validates without architecture or design change.	DDSRV CI
S-DSS-10340	The Document Data Server shall support making stored documents available on physical media within 24 hours of receipt of a Media Distribution Request.	DDSRV CI
S-DSS-10350	The Document Data Server shall be capable of receiving a combined maximum of product orders per hour as derived from Section E.6 of Appendix E (across ECS) from the Data Management Subsystem and/or the Client subsystem.	DDSRV CI
S-DSS-10360	The Document Data Server shall be capable of accepting and storing documents and related data at a nominal rate derived from Section E.1 of Appendix E from external clients while supporting standard data retrieval and access loads.	DDSRV CI
S-DSS-20010	The STMGT CI shall validate all Service Requests.	DsStArchive; DsStResource
S-DSS-20020	The STMGT CI shall accept Insert Requests for insertion of data into the archive.	DsStArchive
S-DSS-20025	The STMGT CI shall place an entry in the Archive Activity Log corresponding to each Insert Request.	DsStArchive
S-DSS-20030	The STMGT CI shall check each Insert Request it receives for the correct type of data in all fields. Fields that shall be checked include Request Identifier, date of request, Priority Information, data type and original identifier.	DsStArchive
S-DSS-20040	The STMGT CI shall accept Retrieve Requests for data. Each Retrieve Request shall include the granule id(s) for the data. Granule id was assigned when granule was originally archived. The granule id serves as a unique data identifier.	DsStArchive
S-DSS-20045	The STMGT CI shall place an entry in the Archive Activity Log corresponding to each Retrieve Request.	DsStArchive
S-DSS-20050	The STMGT CI shall check each Retrieve Request it receives for correct type of data in all fields. Fields that shall be checked include Request Identifier, date of request, date and time for requested data, Priority Information, and data type.	DsStArchive
S-DSS-20060	The STMGT CI shall accept Archive Status Requests for the status of ongoing Insert and Retrieve Requests.	DsStRequestManager

Table A-1. Requirements Trace (18 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-20065	The STMGT CI shall place an entry in the Archive Activity Log corresponding to each Archive Status Request.	DsStRequestManager
S-DSS-20070	The STMGT CI shall check each Archive Status Request it receives for the correct type of data in all fields. Fields that shall be checked include Current Request Identifier and Request Identifier of previous Insert or Retrieve Requests to be statused.	DsStRequestManager
S-DSS-20080	The STMGT CI shall maintain an Archive Activity Log of all Service Requests received. The log of Service Requests shall be in chronological order and shall include a Request Identifier, the operation requested, completion status of request and a date/tim	DsStArchive
S-DSS-20090	The STMGT CI shall maintain an Inventory Update Log. The following information shall be recorded: time and date of update, unique data identifier, archive media name, source of data, storage device name and requester.	DsStArchive
S-DSS-20095	The STGMT CI shall have the capability to mount archival media via automated means.	DsStArchive
S-DSS-20100	The STMGT CI shall provide operations staff personnel the capability to manually access archive media resident in storage devices.	DsStResource
S-DSS-20110	The STMGT CI shall provide operations staff the capability to insert archive media into storage devices which support removable media.	DsStArchive
S-DSS-20120	The STMGT CI shall provide operations staff the capability to remove archive media from storage devices which support removable media.	DsStArchive
S-DSS-20125	The STMGT CI shall, where appropriate, comply with the evolving guidelines and standards emerging from the IEEE Reference Model for Open Storage Systems Interconnection.	STMGT CI
S-DSS-20130	The STMGT CI shall provide operations staff the capability to manually dismount archive media.	DsStArchive
S-DSS-20140	The STMGT CI shall provide operations staff the capability to manually mount archive media.	DsStArchive
S-DSS-20150	The STMGT CI shall provide operations staff the capability to manually dismount backup archive media.	DsStArchive
S-DSS-20160	The STMGT CI shall provide operations staff the capability to manually mount backup archive media.	DsStArchive
S-DSS-20162	The STMGT CI shall provide the capability to mount on-line backup media via automated means.	DsStArchive
S-DSS-20170	The STMGT CI shall automatically request operations staff to load a new archive media to store data if no media exists with sufficient space for the new data.	DsStArchive

Table A-1. Requirements Trace (19 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-20180	The STMGT CI shall have the capability to automatically dismount archive media from storage devices which support removable media when different archive media must be mounted to store data.	DsStArchive
S-DSS-20190	The STMGT CI shall have the capability to automatically dismount archive media from storage devices which support removable media when different archive media must be mounted to retrieve data.	DsStArchive
S-DSS-20200	The STMGT CI shall provide a mechanism to remove archive media from storage devices to allow insertion of new or different archive media in the storage device.	DsStArchive
S-DSS-20220	If an uncorrectable error occurs during archive, the STMGT CI shall notify the operations staff, select a different piece of Media and complete the archive operation. Note: Contents of original media shall be recreated on new media and the original remove	DsStArchive
S-DSS-20230	The STMGT CI shall notify operations staff to discard source archive media after its contents have been re-created on the new media.	DsStArchive
S-DSS-20240	If the end of the archive media is encountered before completing a write operation, the STMGT CI shall select new media and complete the write operation with the new archive media.	DsStArchive
S-DSS-20250	If an uncorrectable error occurs during retrieval operations, STMGT CI shall terminate the operation and notify operations staff and the user/data requester of the failure.	DsStArchive
S-DSS-20255	If an uncorrectable error occurs during retrieval operations, STMGT CI shall automatically recreate the contents on new media.	DsStArchive
S-DSS-20300	The STMGT CI shall provide operations staff the capability to display information about the archive media resident in storage devices. Such information shall include: archive volume name, creation time/ date, archive volume status.	DsStArchive
S-DSS-20350	The STMGT CI shall use a fully described file structure to store data.	DsStArchive
S-DSS-20360	The STMGT CI shall use a fully described physical file organization to store data.	DsStArchive
S-DSS-20370	The STMGT CI shall use openly published and non-proprietary data formats to store data.	DsStArchive
S-DSS-20380	The STMGT CI shall provide the capability to continue operations in a degraded mode despite hardware failures of individual archive storage devices, archive media and/or operator consoles.	DsStArchive
S-DSS-20390	The STMGT CI shall provide operations staff a mechanism for recovery of data as a result of failed archive media. Note: Failed archive media are media which can not be read.	DsStArchive

Table A-1. Requirements Trace (20 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-20400	The STMGT CI shall provide operations staff a mechanism for recovery of data as a result of failed archive storage devices.	DsStArchive
S-DSS-20420	The STMGT CI shall be capable of producing backup archive media which uses openly published and non-proprietary formats for recording data.	DsStArchive
S-DSS-20430	The STMGT CI shall be capable of producing backup archive media which has a fully described file structure.	DsStArchive
S-DSS-20440	The STMGT CI shall be capable of producing backup archive media which has a fully described physical file organization.	DsStArchive
S-DSS-20442	The STMGT CI shall provide the capability to archive Data Availability Schedules.	DsStArchive
S-DSS-20444	The STMGT CI shall provide the capability to retrieve Data Availability Schedules.	DsStArchive
S-DSS-20465	The STMGT CI shall provide the capability to archive non-EOS data required for standard production.	DsStArchive
S-DSS-20475	The STMGT CI shall provide the capability to retrieve non-EOS data to be used for standard product production.	DsStArchive
S-DSS-20480	The STMGT CI shall provide operations staff the capability to perform physical inventories of archive media resident in archive storage devices.	DsStArchive
S-DSS-20490	The STMGT CI shall control access to archived data to prevent unauthorized access.	DsStArchive
S-DSS-20500	The STMGT CI shall report unauthorized attempts to access archived data when detected to operations staff.	DsStArchive
S-DSS-20510	The STMGT CI shall provide operations staff the capability to obtain configuration information about operator selected storage devices.	DsStResource
S-DSS-20520	The STMGT CI shall provide operations staff the capability to change the allocation of storage devices to individual Data Servers.	DsStResourcePolicy
S-DSS-20530	The STMGT CI shall provide the capability to display/ view/print the allocation of storage devices to Data Servers.	DsStResourcePolicy
S-DSS-20540	The STMGT CI shall provide an automatic capability during startup to allocate storage devices to Data Servers.	DsStResource; DsStResource-Config
S-DSS-20590	The STMGT CI shall provide archival storage which is field-expandable. Field-expandable is defined as increasing the capacity or size of archive storage without removing archive storage device from site.	DsStArchive
S-DSS-20600	The STMGT CI shall provide the capability to uniquely identify each data granule that is archived.	DsMdCatalog
S-DSS-20620	The STMGT CI shall provide the capability to retrieve each individual data granule that is stored.	DsStArchive

Table A-1. Requirements Trace (21 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-20650	The STMGT CI shall provide operations staff the capability to generate a backup of all data holdings.	DsStArchive
S-DSS-20660	The STMGT CI shall provide operations staff the capability to restore backups of specified data holdings.	DsStArchive
S-DSS-20670	For each data item archived, the STMGT CI shall record the event in the Inventory Update Log.	DsStArchive
S-DSS-20690	The STMGT CI shall provide the capability to display/ view/print the Inventory Update Log.	DsStArchive
S-DSS-20700	The STMGT CI shall provide the capability to select/ extract Inventory Update Log records for time periods selected by operations staff.	DsStArchive
S-DSS-20710	The STMGT CI shall assign a unique identifier to new archive media.	DsStArchive
S-DSS-20740	The STMGT CI shall provide operations staff the capability to retrieve data that has been safe-stored at an external facility.	DsStArchive
S-DSS-20760	The STMGT CI shall provide operations staff the capability to view/display/print the Archive Activity Log.	DsStStorageResource
S-DSS-20770	The STMGT CI shall provide the capability to sort, extract and/or select Archive Activity Log entries by the following: start/stop time, operation requested, result of request.	DsStStorageResource
S-DSS-20780	The STMGT CI shall provide operations staff the capability to view/display/print the Intermediate Activity Log.	DsStResource
S-DSS-20790	The STMGT CI shall provide the capability to sort, extract and/or select Intermediate Activity Log entries by the following: start/stop time, intermediate operation, Request Identifier, and staging resource(s).	DsStResource
S-DSS-20880	The STMGT CI shall maintain an Intermediate Activity Log. It shall include date/time stamp, operation id (file space alloc./dealloc., media mount/dismount/loads/ unload, file read/write/delete), affiliated Request Identifier and associated staging resource	DsStResource
S-DSS-20890	The STMGT CI shall provide operations staff the capability to load media into storage devices which support removable media.	DsStResource
S-DSS-20900	The STMGT CI shall provide operations staff the capability to initialize media in storage devices which support removable media.	DsStResource
S-DSS-20910	The STMGT CI shall provide operations staff the capability to unload media from storage devices which support removable media.	DsStResource
S-DSS-20980	The STMGT CI shall provide the SDSRV CI the capability to open files on archive storage media in the DRPHW CI.	DsStArchive

Table A-1. Requirements Trace (22 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-20985	The STMGT CI shall provide the SDSRV CI the capability to open files on archive storage media in the WKSHW CI.	DsStStagingFile
S-DSS-20990	The STMGT CI shall provide the SDSRV CI the capability to close files on archive storage media in the DRPHW CI.	DsStArchive
S-DSS-20995	The STMGT CI shall provide the SDSRV CI the capability to close files on archive storage media in the WKSHW CI.	DsStStagingFile
S-DSS-21000	The STMGT CI shall provide the SDSRV CI the capability to read information from files on archive storage media in the DRPHW CI.	DsStArchive
S-DSS-21005	The STMGT CI shall provide the SDSRV CI the capability to read information from files on archive storage media in the WKSHW CI.	DsStStagingFile
S-DSS-21010	The STMGT CI shall provide the SDSRV CI the capability to write information into files on archive storage media in the DRPHW CI.	DsStArchive
S-DSS-21015	The STMGT CI shall provide the SDSRV CI the capability to write information into files on archive storage media in the WKSHW CI.	DsStStagingFile
S-DSS-21020	The STMGT CI shall provide the SDSRV CI the capability to allocate archive storage devices for Service Request processing in the DRPHW CI.	DsStArchive
S-DSS-21025	The STMGT CI shall provide the SDSRV CI the capability to allocate archive storage devices for Service Request processing in the WKSHW CI.	DsStStagingDisk
S-DSS-21030	The STMGT CI shall provide the SDSRV CI the capability to deallocate archive storage devices in the DRPHW CI.	DsStArchive
S-DSS-21035	The STMGT CI shall provide the SDSRV CI the capability to deallocate archive storage devices in the DRPHW CI.	DsStArchive
S-DSS-21040	The STMGT CI shall provide the SDSRV CI the capability to open files on staging devices in the WKSHW CI.	DsStStagingFile
S-DSS-21050	The STMGT CI shall provide the SDSRV CI the capability to close files on staging devices in the WKSHW CI.	DsStStagingFile
S-DSS-21060	The STMGT CI shall provide the SDSRV CI the capability to write information into files on staging devices in the WKSHW CI.	DsStStagingFile
S-DSS-21070	The STMGT CI shall provide the SDSRV CI the capability to read information from files on staging devices in the WKSHW CI.	DsStStagingFile
S-DSS-21080	The STMGT CI shall provide the SDSRV CI the capability to delete files on staging devices in the WKSHW CI.	DsStStagingFile

Table A-1. Requirements Trace (23 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-21080	The STMGT CI shall provide the SDSRV CI the capability to delete files on staging devices in the WKSHW CI.	DsStStagingFile
S-DSS-21090	The STMGT CI shall provide the SDSRV CI the capability to rename files on staging devices in the WKSHW CI.	DsStStagingFile
S-DSS-21100	The STMGT CI shall provide the SDSRV CI the capability to obtain information concerning files on staging devices in the WKSHW CI. Note: File info. includes file name, size, type, organization, creation date, protections, owner, last access time and id of	DsStDirectory
S-DSS-21110	The STMGT CI shall provide the SDSRV CI the capability to allocate storage on staging devices in the WKSHW CI.	DsStStagingDisk
S-DSS-21120	The STMGT CI shall provide the SDSRV CI the capability to deallocate storage on staging devices in the WKSHW CI.	DsStStagingDisk
S-DSS-21160	The STMGT CI shall provide operations staff the capability to set the operational state (UP or DOWN) of storage devices.	DsStStorageResource
S-DSS-21170	The STMGT CI shall provide operations staff the capability to query the operational state (UP or DOWN) of storage devices.	DsStStorageResource
S-DSS-21180	The STMGT CI shall provide operations staff the capability to backup storage system unique files, which shall include all logs, files used by the storage system and files indicating the allocation of storage devices to Data Servers.	DsStStorageResource
S-DSS-21190	The STMGT CI shall provide operations staff the capability to restore storage system unique files.	DsStStorageResource
S-DSS-21200	The STMGT CI shall provide operations staff a mechanism to display/view storage system data storing operations by ECS element.	DsStStorageResource
S-DSS-21210	The STMGT CI shall provide operations staff a mechanism to display/view storage system data retrieval operations by ECS element.	DsStArchive
S-DSS-21220	The STMGT CI shall provide operations staff a mechanism to display/view storage system archive media backup/restore operations by ECS element.	DsStArchive
S-DSS-21230	The STMGT CI shall provide operations staff a mechanism to display/view storage system storage allocations by ECS element.	DsStArchive
S-DSS-21270	The STMGT CI shall provide the operations staff the capability to display information about archive storage devices. Such information shall include current status, current operation, # operations completed, # errors reported, time/date of last error.	DsStArchive

Table A-1. Requirements Trace (24 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-21330	The STMGT CI shall notify operations staff whenever a device failure condition occurs. Such failures shall also be logged in the Archive Activity Log.	DsStResource
S-DSS-21360	The STMGT CI shall use a hierarchy of disk and/or tape storage devices and associated storage media to store data.	DsStArchive
S-DSS-21363	The STMGT CI shall provide location-transparent access to the archived data.	DsStArchive
S-DSS-21365	The STMGT CI shall provide storage for the Data Products listed in Appendix F.	DsStArchive
S-DSS-21366	The STMGT CI shall provide storage for the Metadata associated with the Data Products listed in Appendix F.	DsMdCatalog
S-DSS-21370	The STMGT CI shall use, where appropriate, a hierarchy of disk and/or tape storage devices and associated storage media to retrieve data.	DsStArchive
S-DSS-21380	In the event of storage device or archive media failure, the STMGT CI shall notify operations staff and provide appropriate information to include failed device name or media, failure code or reason and time/date of failure.	DsStArchive
S-DSS-21390	The STMGT CI shall maintain a File Directory of all data files which have been archived and under its control.	DsStFileDirectory
S-DSS-21400	The STMGT CI shall provide operations staff a mechanism to create the File Directory.	DsStFileDirectory
S-DSS-21410	The STMGT CI shall provide operations staff a mechanism to append records to the File Directory.	DsStFileDirectory
S-DSS-21420	The STMGT CI shall provide operations staff a mechanism to display selected records in the File Directory.	DsStFileDirectory
S-DSS-21440	The STMGT CI shall provide operations staff a mechanism to update records in the File Directory.	DsStFileDirectory
S-DSS-21450	The STMGT CI shall provide operations staff the capability to backup the contents of the File Directory.	DsStFileDirectory
S-DSS-21460	The STMGT CI shall provide operations staff the capability to recover the contents of the File Directory in the case of file corruption.	DsStFileDirectory
S-DSS-21470	The STMGT CI shall provide operations staff the capability to view/display/print contents of the File Directory.	DsStFileDirectory
S-DSS-21480	The STMGT CI shall maintain a unique data set id for each data item in its File Directory.	DsStFileDirectory
S-DSS-21490	The STMGT CI shall be capable of tracking the physical location of each data granule via use of the File Directory.	DsStFileDirectory
S-DSS-21500	The Science Management within the Data Server shall support making stored Data Products available on physical media within 24 hours	DSS Subsystem

Table A-1. Requirements Trace (25 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-21510	The Science Management within the Data Server shall be capable of providing of 200% expansion in capacity without architecture or design change.	DSS Subsystem
S-DSS-21520	The Science Management within the Data Server shall be capable of processing a combined maximum number of Data Requests per hour (across ECS) from the Data Management Subsystem and/or the Client Subsystem as derived from Section E.6 of Appendix E.	DSS Subsystem
S-DSS-21540	The Science Management within the Data Server shall support distributing product QA data produced at the collocated Data Processing Subsystem within 1 hour from the time it is ready.	DSS Subsystem
S-DSS-21570	The Science Management within the Data Server shall have the capacity to archive the total bytes of data derived from Section E.1 of Appendix E.	DSS Subsystem
S-DSS-21600	The MSFC DAAC Science Management within the Data Server shall archive original TSDIS standard products (Level 1B-3) after reprocessing for a minimum of 6 months	DSS Subsystem
S-DSS-21630	The GSFC DAAC Science Management within the Data Server shall archive original TSDIS standard products (Level 1B-3) after reprocessing for a minimum of 6 months	DSS Subsystem
S-DSS-21640	The Science Management within the Data Server shall support making archive data associated with a pre-defined ECS standard format available to the network in that format within an average of 2 minutes.	DSS Subsystem
S-DSS-21650	The Science Management within the Data Server shall support making archive data associated with a pre-defined ECS standard format available to the network in a different format within an average of 5 minutes.	DSS Subsystem
S-DSS-21655	The Science Management within the Data Server shall utilize media with a rated shelf life of at least 10 years as determined by National Archives and Record Administration (NARA), National Institute for Standards and Technology (NIST), NASA or an industry	DSS Subsystem
S-DSS-21700	The WKSHW CI shall be sized to temporarily store the number of bytes of data derived from Section E.1 of Appendix E.	WKSHW CI
S-DSS-21710	The WKSHW CI shall be sized to support a sustained I/O rate derived from Section E.1 of Appendix E in bytes/second of data.	WKSHW CI
S-DSS-21720	The DRPHW CI shall be sized to support a sustained I/O rate of 1x the production volume from electronic distribution, where 1x production volume is derived from Section E.1 of, Appendix E.	DRPHW CI
S-DSS-21730	The DRPHW CI shall be sized to permanently store and maintain the total number of bytes of product data derived from Appendix E (Section E.1).	DRPHW CI

Table A-1. Requirements Trace (26 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-21740	The DRPHW CI shall be sized to permanently store and maintain the total number of bytes of record based data derived from Appendix E (Section E.1).	DRPHW CI
S-DSS-21750	The DRPHW CI shall provide a bit error rate after correction less than 1 in 1 X 10**12. (This requirement may be fulfilled with a combination of hardware and software components.)	DRPHW CI
S-DSS-21760	The DRPHW CI shall utilize archive media with a manufactured shelf life of at least 10 years when stored in a controlled environment	DRPHW CI
S-DSS-21770	The DRPHW CI shall be capable of providing of 200 percent expansion in capacity without architecture or design change.	DRPHW CI
S-DSS-21800	The DRPHW CI shall be configured to provide an Availability for archiving data of 0.98000/ and a Mean Down Time (MDT) of <2 hours availability for archiving data.	DRPHW CI
S-DSS-21810	The DRPHW CI shall be configured to provide an Availability for accepting and updating metadata of 0.96000/ and a Mean Down Time (MDT) of <4 hours availability for accepting and updating metadata.	DRPHW CI
S-DSS-21811	The DRPHW CI shall be configured to provide an Availability for information searches in the ECS Directory of 0.993/ and a Mean Down Time (MDT) of <2 hours availability for information searches on the ECS directory.	DRPHW CI
S-DSS-21813	The DRPHW CI shall be configured to provide an Availability for searches on local holdings of 0.96/ and a Mean Down Time (MDT) of <4 hours availability for searches on local holdings.	DRPHW CI
S-DSS-21814	The DRPHW CI shall be configured to provide an Availability for database management and maintenance interface functions of 0.96/ and a Mean Down Time (MDT) of <4 hours availability for database management and maintenance interface functions.	DRPHW CI
S-DSS-21815	The DRPHW CI shall support the RMA Requirements specified for the ECS Program.	DRPHW CI
S-DSS-30010	The DDIST CI shall accept Electronic Distribution Requests or Media Distribution Requests.	DsDdDistRequest
S-DSS-30020	Distribution Requests shall have the format described in Appendix A.	DsDdDistRequest
S-DSS-30030	The DDIST CI shall validate each Electronic Distribution Request and verify that the format conforms to that specified in Appendix A.	DsDdDistRequest
S-DSS-30040	The DDIST CI shall log the following on the Distribution Activity Log whenever an Electronic Distribution Request fails validation: User Identifier, Request Identifier, Date and Time, and an explanation of the failure.	DsDdRequestProcessor

Table A-1. Requirements Trace (27 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-30045	The DDIST CI shall send Notifications to users via email in the event that the request is canceled by operations staff and the user has an active session.	DsDdRequestProcessor
S-DSS-30046	The DDIST CI shall send Notifications to the user's desktop application in the event that the request is canceled by operations staff and the user has an active session.	DsDdRequestProcessor
S-DSS-30050	The DDIST CI shall send a Notification to the source of the request if an Electronic Distribution Request fails validation.	DsDdRequestProcessor
S-DSS-30060	The DDIST CI shall validate each Media Distribution Request and verify that it conforms to the format specified in Appendix A.	DsDdDistRequest
S-DSS-30070	The DDIST CI shall log a Distribution Failure Message whenever a Media Distribution Request fails validation.	DsDdRequestProcessor
S-DSS-30080	The DDIST CI shall send a Notification to the source of the request if a Media Distribution Request fails validation.	DsDdRequestProcessor
S-DSS-30090	The DDIST CI shall provide the capability to prioritize requests for data based on whether the request is an Electronic Distribution Request or a Media Distribution Request.	DsDdRequestManager
S-DSS-30100	The DDIST CI shall provide operations staff the capability to change the Priority Information for a Distribution Request before the processing of the request has begun.	DsDdPrivRequest
S-DSS-30110	The DDIST CI shall provide the capability for operations staff to list Distribution Requests according to whether the request is an Electronic Distribution Request or a Media Distribution Request.	DsDdRequestList
S-DSS-30115	The DDIST CI shall provide the capability for operations staff to list Distribution Requests according to Request Identifier and status.	DsDdRequestList
S-DSS-30120	The DDIST CI shall provide the capability for operations staff to select for viewing Media Distribution Requests and Electronic Distribution Requests.	DsDdRequestList
S-DSS-30130	The DDIST CI shall provide the capability for operations staff to cancel the processing of Electronic Distribution Requests prior to the start of the transmission of the data.	DsDdDistRequest
S-DSS-30140	The DDIST CI shall provide the capability for operations staff to cancel the data transmission initiated by the processing of an Electronic Distribution Request.	DsDdDistRequest
S-DSS-30150	The DDIST CI shall provide the capability for operations staff to cancel the processing of a Media Distribution Request prior to the shipment of the media.	DsDdDistRequest

Table A-1. Requirements Trace (28 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-30160	The DDIST CI shall send a Notification to the originator of a Distribution Request in the event that the request is canceled by operations staff.	DsDdRequestProcessor
S-DSS-30163	The DDIST CI shall authenticate the User Identifier of operations staff submitting an Ingest Cancellation Request.	DsDdDistRequest
S-DSS-30165	The DDIST CI shall log a Distribution Request Cancellation Message whenever a Distribution Request is cancelled by the operations staff.	DsDdRequestProcessor
S-DSS-30167	The DDIST CI shall log a Distribution Request Cancellation Message whenever a Distribution Request is cancelled by the science user.	DsDdRequestProcessor
S-DSS-30170	The DDIST CI shall respond to Status Requests from science users with a Request State indicating that the specified Distribution Request is “pending”, “active”, or “not found”.	DsDdDistRequestC
S-DSS-30171	The DDIST CI shall respond to Status Requests from operations staff with a Request State indicating that the specified Distribution Request is “pending”, “staging”, “transferring” or “not found”.	DsDdOpsRequestC
S-DSS-30175	Status Requests shall have the format given in Appendix A.	DsDdDistRequest
S-DSS-30180	The DDIST CI shall process queued Distribution Requests in prioritized order.	DsDdRequestManager
S-DSS-30245 *	The DDIST CI shall provide the capability to report accounting data to the SDSRV CI.	DsAdRequestInterface
S-DSS-30250	Upon the receipt of a status request, DDIST shall validate and provide the status of previously submitted distribution request.	DsDdDistRequest
S-DSS-30260	The DDIST CI shall log the receipt of a Data Distribution Request in the Distribution Activity Log.	DsDdRequestManager
S-DSS-30270	The DDIST CI shall log the following to the Distribution Activity Log, for each Media Distribution Request: User Identifier, Media Identifiers, Media Type/Form Factor, and the Distribution Size.	DsDdRequestProcessor
S-DSS-30280	The DDIST CI shall log the following to the Distribution Activity Log, for each Electronic Distribution Request: User Identifier, Data Destination, and the Distribution Size.	DsDdRequestProcessor
S-DSS-30288	The DDIST CI shall forward the Distribution Activity Log entries to the SMC.	DsDdDistRequest
S-DSS-30290	The DDIST CI shall provide operations staff with the capability to display the Distribution Activity Log.	DsDdDistRequest
S-DSS-30295	The DDIST CI shall alert operations staff when electronic transmission problems are encountered.	DsDdPushProcessor
S-DSS-30300	The DDIST CI shall provide the capability to view entries according to type of distribution, by time period or by Request Identifier (i.e., source of request).	DsDdRequestList

Table A-1. Requirements Trace (29 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-30305	The DDIST CI shall provide the capability to view entries according to type of distribution, by time period or by data type (i.e., source of request).	DsDdRequestList
S-DSS-30310	The DDIST CI shall provide the capability to sort the Distribution Activity Log by distribution type (i.e., electronic (push/pull) and physical media type (tape, CD-ROM, etc.)).	DsDdDistRequest
S-DSS-30320	The DDIST CI shall record in the Distribution Activity Log the occurrence of correctable errors.	DsDdRequestProcessor
S-DSS-30330	If the DDIST CI is unable to distribute data electronically, the User Identifier, the list of data, and the reason for the failure will be logged.	DsDdPushProcessor
S-DSS-30340	If the DDIST CI is unable to distribute data electronically, the user shall be sent a Notification.	DsDdPushProcessor
S-DSS-30350	The DDIST CI shall provide the capability to generate reports on the distribution activity for a period specified by operations staff.	DsDdRequestManager
S-DSS-30355	The DDIST CI shall provide the capability to generate reports on the distribution backlog.	DsDdRequestList
S-DSS-30370	The DDIST CI shall log the number of physical media that is created during distribution.	DsDdRequestProcessor
S-DSS-30380	The DDIST CI shall log the Media Destination and the number of data items distributed in a physical media distribution.	DsDdRequestProcessor
S-DSS-30390	The DDIST CI shall log the Data Destination and the number of data items distributed in an electronic distribution.	DsDdPushProcessor
S-DSS-30400	The DDIST CI shall log the User Identifier for the user that originated the Data Distribution Request.	DsDdRequestProcessor
S-DSS-30430	The DDIST CI shall provide the capability for the operations staff to manually enter the status of a physical media shipment. Status will be updated from “waiting for shipment” to “shipped”.	DsDdDistRequest
S-DSS-30431	The DDIST CI shall log a physical media shipment using the following categories: pending, active, waiting for shipment.	DsDdRequestProcessor
S-DSS-30440	The DDIST CI shall provide the capability to distribute on 8mm tape.	DsDdTapeProcessor
S-DSS-30470	The DDIST CI shall provide the capability to distribute on CD ROM.	DsDdCDProcessor
S-DSS-30480	The DDIST CI shall provide the capability to distribute on 6250 tape.	DsDdTapeProcessor
S-DSS-30490	If an uncorrectable error occurs while writing to distribution media, the operation shall be aborted and a new piece of media automatically requested from operations staff.	DsDdRequestProcessor

Table A-1. Requirements Trace (30 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-30520	The DDIST CI shall provide the capability to place Data in publicly available disks for users to “pull” the data, via ftp, at their discretion.	DsDdPullProcessor
S-DSS-30530	The DDIST CI shall provide the capability to limit access to Data in the user pull area to the science user and the operations staff.	DsStPullList
S-DSS-30540	The DDIST CI shall monitor the percentage of space utilized in the user pull area.	DsStPullMonitor
S-DSS-30550	The DDIST CI shall provide a mechanism for operations staff to view/display the percentage of space utilized in the user pull area.	DsStPullMonitor
S-DSS-30560	The DDIST CI shall notify operations staff if the percent utilization in the user pull area exceeds a specified threshold.	DsStPullMonitor
S-DSS-30570	When Data is placed in the user pull area, requesting user shall be notified that the Data is available for a limited time.	DsDdPullProcessor
S-DSS-30575	The DDIST CI shall notify operations staff when the time limit has expired for Data in the user pull area.	DsStPullMonitor
S-DSS-30580	The DDIST CI shall, after operator confirmation, delete expired Data from the user pull area.	DsStPullMonitor
S-DSS-30585	Operations staff shall be able to turn off the function of operator confirmation associated with the automatic deletion of Data in the user pull area.	DsStPullConfig
S-DSS-30600	The DDIST CI shall provide the capability to distribute Data electronically via ftp (push).	DsDdPushMedia
S-DSS-30640	The DDIST CI shall provide the capability for operations staff to change the state (on-line vs. off-line) of a peripheral device that is used for distribution.	DsStResource
S-DSS-30650	The DDIST CI shall provide the capability for operations staff to display the state (on-line vs. off-line) of peripheral distribution devices.	DsStResource
S-DSS-30660	In the event of media failure (i.e., tape breaks), the DDIST CI shall provide the capability to restart the distribution on a new piece of media.	DsDdMedia
S-DSS-30670	If an electronic push distribution fails, DDIST CI shall make a system defined number of additional attempts before aborting the transmission and notifying the originator of the failure. These additional attempts shall be included in the Distribution Act	DsDdPushProcessor
S-DSS-30680	The DDIST CI shall provide the capability for operations staff to change the system defined number of additional attempts for re-transmission.	DsDdPushProcessor
S-DSS-30705	For physical media distributions, DDIST CI shall generate a packing list describing the data on the media.	DsDdPackingSlip

Table A-1. Requirements Trace (31 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-30710	The DDIST CI shall provide the capability to distribute any Data, or appropriate subset, listed in the Inventory. Note: The appropriate subset of a data item is determined by and depends on the subject data type.	DsDsDistRequest
S-DSS-30730	The DDIST CI shall provide the capability for operations staff to manually load media into the peripheral devices.	DsStResource
S-DSS-30740	The DDIST CI shall provide the capability for operations staff to manually unload media from the peripheral devices.	DsStResource
S-DSS-30750	The DDIST CI shall provide the capability for the operations staff to specify a percent utilization threshold for the user pull area above which operations staff will be notified.	DsStPullConfig
S-DSS-30780	The DDIST CI shall make appropriate use of standards for data structures and data transport as defined for use within the publications of CCSDS and ISO/OSI for distribution of TRMM data to the TSDIS.	DsDdDistRequest
S-DSS-30797	The DDIST CI shall receive management directives from the SDSRV CI.	DsDdDistRequestC
S-DSS-30800	The Data Distribution within the Data Server shall support making stored products available on physical media within 24 hours.	DDIST CI
S-DSS-30810	The Data Distribution within the Data Server shall be capable of distributing Data via physical media generated a rate equivalent to the daily rate data are ingested at that site.	DDIST CI
S-DSS-30840	The Data Distribution within the Data Server shall support distributing product QA data produced at the collocated Data Processing Subsystem within 1 hour from the time it is ready.	DDIST CI
S-DSS-30850	The Data Distribution within the Data Server shall support making archive data associated with a predefined ECS standard format available to the network in that format within an average of 2 minutes.	DDIST CI
S-DSS-30860	The Data Distribution within the Data Server shall support making archive data associated with a predefined ECS format available to the network in a different format within an average of 5 minutes.	DDIST CI
S-DSS-30870	The DAAC Data Distribution within the Data Server shall be capable of electronically distributing data to users in support of Electronic Distribution Requests at a rate equivalent to daily product volume, L1-L4.	SDSRV CI
S-DSS-30875	The Data Distribution within the Data Server shall be capable of providing 200% expansion in capacity without architecture or design change.	SDSRV CI
S-DSS-30890	The Data Distribution within the Data Server shall be capable of distributing the number of bytes of data per day derived from Section E.6 of Appendix E to TSDIS (for the purpose of reprocessing).	SDSRV CI

Table A-1. Requirements Trace (32 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-30950	The DIPHW CI shall be sized to temporarily store the total number of bytes of distribution data derived from Section E.1 of Appendix E.	DIPHW CI
S-DSS-30960	The DIPHW CI shall be sized to support a sustained I/O rate of 1x the production volume for media distribution, where 1x production volume is derived from Section E.1 of, Appendix E.	DIPHW CI
S-DSS-31000	The DIPHW CI shall be configured to support the SDPS function of Archiving and Distributing data Availability requirement of 0.98000 and a Mean Down Time requirement of <2 hours. (This applies to distributing data and ingesting hard media.)	DIPHW CI
S-DSS-31005	The DIPHW CI shall support the RMA Requirements specified for the ECS Program.	DIPHW CI
S-DSS-60010	The electrical power requirements for ACMHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	ACMHW CI
S-DSS-60020	The air conditioning requirements for the ACMHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	ACMHW CI
S-DSS-60030	The grounding requirements for ACMHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ACMHW CI
S-DSS-60040	The fire alarm requirements for ACMHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ACMHW CI
S-DSS-60050	The acoustical requirements for ACMHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ACMHW CI
S-DSS-60060	The physical interface requirements between ACMHW CI equipment and the facility shall be in accordance with ECS Facilities Plan (DID 302/DV2).	ACMHW CI
S-DSS-60070	The footprint size and the physical layout of ACMHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	ACMHW CI
S-DSS-60110	The operating system for each Unix platform in the ACMHW CI shall conform to the POSIX.2 standard.	ACMHW CI
S-DSS-60120	The ACMHW CI POSIX.2 compliant platform shall have the following utilities installed at a minimum: perl, emacs, gzip, tar, imake, prof, gprof, nm.	ACMHW CI
S-DSS-60130	The ACMHW CI POSIX.2 compliant platform shall have the following POSIX.2 user Portability Utilities installed at a minimum: man, vi.	ACMHW CI
S-DSS-60140	The ACMHW CI POSIX.2 compliant platform shall have the following POSIX.2 Software Development Utilities installed at a minimum: make.	ACMHW CI
S-DSS-60150	The ACMHW CI POSIX.2 compliant platform shall have the following POSIX.2 C-Language Development Utilities installed at a minimum: lex, yacc.	ACMHW CI

Table A-1. Requirements Trace (33 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-60160	The ACMHW CI POSIX.2 compliant platform shall have the following Unix shells installed at a minimum: C shell, Bourne shell, Korn shell.	ACMHW CI
S-DSS-60170	The ACMHW CI POSIX.2 compliant platform shall have on-line documentation or printed documentation for each installed tool.	ACMHW CI
S-DSS-60180	The ACMHW CI POSIX.2 compliant platform shall have installed one or more development environment supporting the following languages:	ACMHW CI
S-DSS-60190	Each development environment associated with the POSIX.2 compliant platform in the ACMHW CI shall have the capability to compile and link strictly conformant POSIX-compliant source code.	ACMHW CI
S-DSS-60195	Each development environment associated with the POSIX.2 compliant platform in the ACMHW CI shall have an interactive source level debugger for ECS supported languages.	ACMHW CI
S-DSS-60930	The ACMHW CI at the GSFC DAAC shall be capable of ingesting data from TSDIS at the nominal rate specified in Sections E.2 & E.3 of Appendix E.	ACMHW CI
S-DSS-60940	The ACMHW CI at the GSFC DAAC shall be capable of ingesting data at a maximum rate that is three times the nominal rate specified in Sections E.2 & E.3 of Appendix E.	ACMHW CI
S-DSS-60950	The ACMHW CI at the GSFC DAAC shall be capable of ingesting Version 0 data at the nominal rate specified in Section E.4 of Appendix E.	ACMHW CI
S-DSS-60970	The ACMHW CI at the GSFC DAAC shall be capable of ingesting data at a nominal rate of TBD bytes per day from the DAO by network data transfer.	ACMHW CI
S-DSS-61010	The ACMHW CI at the LaRC DAAC shall be capable of ingesting Version 0 data by network data transfer at the nominal rate specified in Section E.4 of Appendix E.	ACMHW CI
S-DSS-61020	The ACMHW CI at the LaRC DAAC shall be capable of ingesting data at a nominal rate of TBD bytes per day from the DAO by network data transfer.	ACMHW CI
S-DSS-70010	The electrical power requirements for WKSHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	WKSHW CI
S-DSS-70020	The air conditioning requirements for the WKSHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	WKSHW CI
S-DSS-70030	The grounding requirements for WKSHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	WKSHW CI
S-DSS-70040	The fire alarm requirements for WKSHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	WKSHW CI

Table A-1. Requirements Trace (34 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-70050	The acoustical requirements for WKSHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	WKSHW CI
S-DSS-70060	The physical interface requirements between WKSHW CI equipment and the facility shall be in accordance with ECS Facilities Plan (DID 302/DV2).	WKSHW CI
S-DSS-70070	The footprint size and the physical layout of WKSHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	WKSHW CI
S-DSS-70075	The WKSHW CI shall support the RMA Requirements specified for the ECS Program.	WKSHW CI
S-DSS-80010	The electrical power requirements for DRPHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	DRPHW CI
S-DSS-80020	The air conditioning requirements for the DRPHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	DRPHW CI
S-DSS-80030	The grounding requirements for DRPHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DRPHW CI
S-DSS-80040	The fire alarm requirements for DRPHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DRPHW CI
S-DSS-80050	The acoustical requirements for DRPHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DRPHW CI
S-DSS-80060	The physical interface requirements between DRPHW CI equipment and the facility shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DRPHW CI
S-DSS-80070	The footprint size and the physical layout of DRPHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	DRPHW CI
S-DSS-80110	The operating system for each Unix platform in the DRPHW CI shall conform to the POSIX.2 standard.	DRPHW CI
S-DSS-80120	The DRPHW CI POSIX.2 compliant platform shall have the following utilities installed at a minimum: perl, emacs, gzip, tar, imake, prof, gprof, nm.	DRPHW CI
S-DSS-80130	The DRPHW CI POSIX.2 compliant platform shall have the following POSIX.2 user Portability Utilities installed at a minimum: man, vi.	DRPHW CI
S-DSS-80140	The DRPHW CI POSIX.2 compliant platform shall have the following POSIX.2 Software Development Utilities installed at a minimum: make.	DRPHW CI
S-DSS-80150	The DRPHW CI POSIX.2 compliant platform shall have the following POSIX.2 C-Language Development Utilities installed at a minimum: lex, yacc.	DRPHW CI

Table A-1. Requirements Trace (35 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-80160	The DRPHW CI POSIX.2 compliant platform shall have the following Unix shells installed at a minimum: C shell, Bourne shell, Korn shell.	DRPHW CI
S-DSS-80170	The DRPHW CI POSIX.2 compliant platform shall have on-line documentation or printed documentation for each installed tool.	DRPHW CI
S-DSS-80180	The DRPHW CI POSIX.2 compliant platform shall have installed one or more development environment supporting the following languages:	DRPHW CI
S-DSS-80190	Each development environment associated with the POSIX.2 compliant platform in the DRPHW CI shall have the capability to compile and link strictly conformant POSIX-compliant source code.	DRPHW CI
S-DSS-80195	Each development environment associated with the POSIX.2 compliant platform in the DRPHW CI shall have an interactive source level debugger for ECS supported languages.	DRPHW CI
S-DSS-90010	The electrical power requirements for DIPHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	DIPHW CI
S-DSS-90020	The air conditioning requirements for the DIPHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	DIPHW CI
S-DSS-90030	The grounding requirements for DIPHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DIPHW CI
S-DSS-90040	The fire alarm requirements for DIPHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DIPHW CI
S-DSS-90050	The acoustical requirements for DIPHW CI equipment shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DIPHW CI
S-DSS-90060	The physical interface requirements between DIPHW CI equipment and the facility shall be in accordance with ECS Facilities Plan (DID 302/DV2).	DIPHW CI
S-DSS-90070	The footprint size and the physical layout of DIPHW CI equipment shall be in accordance with the ECS Facilities Plan (DID 302/DV2).	DIPHW CI
S-DSS-90110	The operating system for each Unix platform in the DIPHW CI shall conform to the POSIX.2 standard.	DIPHW CI
S-DSS-90120	The DIPHW CI POSIX.2 compliant platform shall have the following utilities installed at a minimum: perl, emacs, gzip, tar, imake, prof, gprof, nm.	DIPHW CI
S-DSS-90130	The DIPHW CI POSIX.2 compliant platform shall have the following POSIX.2 user Portability Utilities installed at a minimum: man, vi.	DIPHW CI
S-DSS-90140	The DIPHW CI POSIX.2 compliant platform shall have the following POSIX.2 Software Development Utilities installed at a minimum: make.	DIPHW CI

Table A-1. Requirements Trace (36 of 36)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-90150	The DIPHW CI POSIX.2 compliant platform shall have the following POSIX.2 C-Language Development Utilities installed at a minimum: lex, yacc.	DIPHW CI
S-DSS-90160	The DIPHW CI POSIX.2 compliant platform shall have the following Unix shells installed at a minimum: C shell, Bourne shell, Korn shell.	DIPHW CI
S-DSS-90170	The DIPHW CI POSIX.2 compliant platform shall have on-line documentation or printed documentation for each installed tool.	DIPHW CI
S-DSS-90180	The DIPHW CI POSIX.2 compliant platform shall have installed one or more development environment supporting the following languages:	DIPHW CI
S-DSS-90190	Each development environment associated with the POSIX.2 compliant platform in the DIPHW CI shall have the capability to compile and link strictly conformant POSIX-compliant source code.	DIPHW CI
S-DSS-90195	Each development environment associated with the POSIX.2 compliant platform in the DIPHW CI shall have an interactive source level debugger for ECS supported languages.	DIPHW CI
S-DSS-90300	The DIPHW CI at the GSFC DAAC shall be capable of ingesting Version 0 data from physical media agreed upon between ECS and Version 0, at the nominal rate specified in Section E.45 of Appendix E.	DIPHW CI
S-DSS-90310	The DIPHW CI at the MSFC DAAC shall be capable of ingesting Version 0 data from physical media agreed upon between ECS and Version 0, at the nominal rate specified in Section E.4 of Appendix E.	DIPHW CI
S-DSS-90320	The DIPHW CI at the LARC DAAC shall be capable of ingesting Version 0 data from physical media agreed upon between ECS and Version 0, at the nominal rate specified in Section E.4 of Appendix E.	DIPHW CI

* Fulfilled in Release B

Abbreviations and Acronyms

ADSRV	advertising service CSCI
AHWGP	Ad Hoc Working Group for Production
AMASS	archival management and storage system
APC	Access/Process Coordinators
API	application program interface
AS	Administration Stations
ASCII	American Standard Code for Information Interchange
ATL	automatic tape library
ATM	asynchronous transmission mode
CD-ROM	compact disk - read only memory
CGI	common gateway interface
CI	configuration item
COTS	commercial off-the-shelf
CPU	central processing unit
CSC	computer software component
CSCI	computer software configuration item
CSS	Communication Services Subsystem
DAAC	distributed active archive center
DAS	Dual-Attached Station
DBA	database administrator
DBMS	database management system
DCE	distributed computing environment
DDICT	data dictionary CSCI
DDIST	data distribution CSCI
DDSRV	document data server CSCI
DEV	developed code
DLT	Digital Linear Tape
DPS	data processing subsystem
DSS	data server subsystem
ECS	EOSDIS Core System
EDF	ECS development facility
EDHS	Electronic Document Handling System

EOSDIS	Earth Observing System Data and Information System
ESDT	Earth science data types
FDDI	fiber distributed data interface
FSMS	File Storage Management System
FTP	file transfer protocol
GUI	graphical user interface
HSM	hierarchical storage management
HTML	hypertext markup language
HTTP	hypertext transfer protocol
HW	hardware
I/O	input/output
INGST	Ingest Subsystem
INTOP	Interoperability CSCI
MSS	management subsystem
NTP	New Tape Product (IBM designation of the 3590 tape)
OMT	Object Modeling Technique
OODCE	object oriented DCE
OPS	operations
OTS	off-the-shelf
PAM	Permanent Archive Management
pdf	portable document format
RAID	redundant array of inexpensive disks
RMA	reliability, maintainability, availability
RTF	Relocatable Transfer Format
SAS	Single-Attached Station
SDPS	Science Data Processing Segment
SDSRV	Science Data Server CSCI
SMP	symmetric multi-processor
SNMP	simple network management protocol
STMGT	Storage Management Software CSCI
TRMM	Tropical Rainfall Measuring Mission
TSDIS	TRMM Science Data and Information System
V0	Version 0
V1	Version 1

WAIS	Wide Area Information Service
WS	working storage
WWW	World Wide Web