HUGHES INFORMATION TECHNOLOGY SYSTEMS

ERRATA NOTICE

EOS Core System (ECS) Project Contract No. NAS5-60000

June 12, 1996

Document No.: 430-TP-008-001

Title: Change Pages for Release B SDPS Data Server Subsystem Design Specification for the ECS Project

Enclosed please find change pages for the subject document. Please replace the pages as follows:

Remove	Insert
4-735 and 4-736	4-735 and 4-736
4-749 and 4-750	4-749 and 4-750
4-753 and 4-754	4-753 and 4-754

If you have any questions, please contact our Data Management Office at (301) 925-0313.

430-TP-008-001

Change Pages for Release B SDPS Data Server Subsystem Design Specification for the ECS Project

Technical paper—ECS Project only — Not for Distribution

June 1996

Prepared Under Contract NAS5-60000

SUBMITTED BY

Rich Kochhar /s/ 6/6/96

Rick Kochhar, Release B CCB Chairman Date
EOSDIS Core System Project

Hughes Information Technology Systems
Upper Marlboro, Maryland

This page intentionally left blank.

Preface

This technical paper provides change pages for the Release B SDPS Data Server Subsystem Design Specification (305-CD-024-002). The change pages provide updated detail design material for the Science Data Server (SDSRV) and Document Data Server (DDSRV) Computer Software Configuration Items (CSCI). In addition to the detail design sections for these two CSCI, other sections of the document have been updated as necessary to reflect changes associated with these two CSCIs. The sections provided replace the corresponding sections in CDRL 305-CD-024-002. The complete Release B SDPS Data Server Subsystem Design Specification will be reissued as an "As Built" version consistent with the ECS contract.

Object models presented in this document have been exported directly from CASE or DBMS 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 (EDHS) at: URL http://edhs1.gsfc.nasa.gov.

This technical paper contains changes to a formal contract deliverable document with an approval code of 2; as such it requires Government review and approval prior to acceptance and use. This document is under ECS contractor configuration control. Once this document is approved, Contractor approved changes 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 Systems
1616 McCormick Drive
Upper Marlboro, MD 20774-5372

This page intentionally left blank.

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.

Contents

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

3. Subsystem Overview

3.1 Introduction and Context 3-1
3.2 Subsystem Overview 3-8
3.2.1 Subsystem Structure 3-8
3.2.2. Subsystem Design Rationale 3-12
3.2.3. Data Server Subsystem Common Design 3-18
3.2.4. Data Server Subsystem Use of Key Design Mechanisms 3-19

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 ac Class Category 4-7
4.3.2 ad Class Category 4-9
4.3.3 as Class Category 4-16
4.3.4 av Class Category 4-18
4.3.5 ba Class Category 4-20
4.3.6 ce Class Category 4-22
4.3.7 cl Class Category 4-26
4.3.8 cn Class Category 4-86
4.3.9 co Class Category 4-97
4.3.10 cs Class Category 4-99
4.3.11 da Class Category 4-145
4.3.12 db Class Category 4-147
4.3.13 de Class Category 4-163
4.3.14 dm Class Category 4-271
4.3.15 er Class Category 4-273
4.3.16 et Class Category 4-275

4.3.17 ge Class Category 4-277
4.3.18 gl Class Category 4-319
4.3.19 gu Class Category 4-481
4.3.20 gv Class Category 4-487
4.3.21 je Class Category 4-489
4.3.22 md Class Category 4-491
4.3.23 mo Class Category 4-539
4.3.24 mp Class Category 4-541
4.3.25 ms Class Category 4-543
4.3.26 nm Class Category 4-545
4.3.27 np Class Category 4-547
4.3.28 ns Class Category 4-552
4.3.29 pr Class Category 4-561
4.3.30 ra Class Category 4-563
4.3.31 sa Class Category 4-565
4.3.32 sh Class Category 4-567
4.3.33 sr Class Category 4-647
4.3.34 sw Class Category 4-694
4.3.35 tm Class Category 4-696
4.3.36 vi Class Category 4-698
4.4 CSCI Dynamic Model 4-700
4.5 Return Request Status 4-702
4.4.1 SDSRV Acquiring An ESDT 4-704
4.4.2 SDSRV Aster On Demand Process Request 4-706
4.4.3 SDSRV Asynchronous Status Updates 4-708
4.4.4 SDSRV Catalog Deleting A Metadata Entry 4-710
4.4.5 SDSRV Catalog Insert Collection Metadata 4-712
4.4.6 SDSRV Catalog Insertion of Metadata 4-714
4.4.7 SDSRV Catalog Search 4-716
4.4.8 SDSRV Catalog Updating Metadata 4-718
4.4.9 SDSRV Changing A Request Priority 4-720
4.4.10 SDSRV Client Browsing 4-722
4.4.11 SDSRV Client Connecting to a Data Server 4-724
4.4.12 SDSRV Client Request Submission 4-726
4.4.13 SDSRV Client Searching 4-728
4.4.14 SDSRV Deleting A Queued Request 4-730
4.4.15 SDSRV Inserting Composite ESDT 4-732
4.4.16 SDSRV Inserting New ESDT 4-734
4.4.17 SDSRV Inserting Single ESDT 4-736

4.4.18 SDSRV Instantiating An ESDT 4-738
4.4.19 SDSRV Multi State Granule 4-740
4.4.20 SDSRV Op View Queued Requests 4-742
4.4.21 SDSRV Server Handling A Browse Request 4-744
4.4.22 SDSRV Server Handling A Search Request 4-746
4.4.23 SDSRV Server Request Handling 4-748
4.4.24 SDSRV Startup Of A Science Data server 4-750
4.4.25 SDSRV Subsetting an ESDT 4-752
4.4.26 SDSRV Suspend and Resume 1 Suspend Initial Closing 4-755
4.4.27 SDSRV Suspend and Resume 2 Suspend Client Side 4-758
4.4.28 SDSRV Suspend and Resume 3 Suspend Server Side 4-760
4.4.29 SDSRV Suspend and Resume 4 Resume Session Client Side 4-762
4.4.30 SDSRV Suspend and Resume 5 Resuming Session Server Side 4-764
4.4.31 SDSRV Suspend and Resume 6 Reconstructing A Ge Suspended Object 4-766
4.4.32 SDSRV Suspend and Resume 7 Restoring Server Side Session 4-768
4.4.33 SDSRV Suspend and Resume 8 Restoring Client Side Session State 4-770
4.4.34 SDSRV Update Server Configuration 4-772
4.4.35 SDSRV Validating Metadata 4-774
4.5 CSCI Structure 4-776
4.5.1 CSC Definitions 4-777
4.5.2 CSCI Dynamic Architecture 4-791
4.6 CSCI Management and Operation 4-793
4.6.1 System Management Strategy 4-793
4.6.2 Operator Interfaces 4-796
4.6.3 Standard SDSRV Reports 4-797

5. DDSRV - Document Data Server CSCI

5.1 CSCI Overview 5-1
5.2 CSCI Context 5-1
5.3 CSCI Object Model 5-3
5.3.1 cd Class Category 5-5
5.3.2 ct Class Category 5-8
5.3.3 do Class Category 5-23
5.3.4 es Class Category 5-38
5.3.5 gw Class Category 5-54
5.3.6 se Class Category 5-67
5.3.7 sv Class Category 5-69

5.4 CSCI Dynamic Model 5-72
5.4.1 DDSRV Document Insert 5-73
5.4.2 DDSRV search 5-75
5.4.3 DDSRV Acquire Dynamic Model 5-77
5.4.4 DDSRV Fault Scenario Dynamic Model 5-79
5.5 CSCI Structure 5-81
5.5.1 DDSRV CSCs 5-81
5.5.2 DDSRV CI Processes 5-84
5.6 CSCI Management and Operation 5-85
5.6.1 System Management Strategy 5-85
5.6.2 Operator Interfaces 5-86
5.6.3 Standard DDSRV Reports 5-88

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 in Release B 10-2
10.2 HWCI Structure 10-3
10.2.1 HWCI Connectivity 10-4
10.2.2 HWCI Component Description 10-6
10.2.3 Failover and Recovery Strategy 10-6

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 Future Releases 12-3
12.2 HWCI Structure 12-3
12.2.1 HWCI Connectivity 12-3
12.2.2 HWCI Components 12-4
12.2.3 Failover and Recovery Strategy 12-5

Appendix A. Requirements Trace

List of Figures

3.2.2-1 Encapsulation of COTS Products in the Data Server Subsystem 3-13

4.2-1 SDSRV CI Interfaces 4-2
1 ac Class Main 4-7
2 ad Class Main 4-9
3 as Class Main 4-16
4 av Class Main 4-18
5 ba Class Main 4-20
6 ce Class Main 4-22
7 cl Class Main 4-26
8 DsCl2 4-27
9 Main 4-28
10 cn Class Main 4-86
11 co Class Main 4-97
12 DsCs1 4-99
13 DsCs2 4-100
14 Main 4-101
15 da Class Main 4-145
16 db Class Main 4-147
17 DsDe1 4-163
18 DsDe2 4-164
19 DsDe3 4-165
20 DsDe4 4-166
21 DsDe5 4-167
22 DsDe6 4-168
23 dm Class Main 4-271
24 er Class Main 4-273
25 et Class Main 4-275
26 DsGe1 4-278
27 DsGe2 4-279
28 Gl0 4-320
29 Gl1 4-321
30 Gl2 4-322
31 Gl3 4-323
32 Gl4 4-324
33 gu Class Main 4-481
34 gv Class Main 4-487
35 je Class Main 4-489
36 md Class Main 4-492
37 mo Class Main 4-539
38 mp Class Main 4-541

39 ms Class Main 4-543
40 nm Class Main 4-545
41 np Class Main 4-547
42 ns Class Main 4-553
43 pr Class Main 4-561
44 ra Class Main 4-563
45 sa Class Main 4-565
46 DsSh1 4-568
47 DsSh2 4-569
48 DsSh3 4-570
49 DsSh4 4-571
50 DsSh5 4-572
51 DsSh6 4-573
52 DsSr1 4-647
53 DsSr2 4-648
54 DsSr3 4-649
55 DsSr4 4-650
56 sw Class Main 4-694
57 tm Class Main 4-696
58 vi Class Main 4-698
59 Return Request Status 4-703
60 SDSRV Acquiring and ESDT 4-705
61 SDSRV Aster On Demand Process Request 4-707
62 SDSRV Asynchronous Status Updates 4-709
63 SDSRV Catalog Deleting A Metatdata Entry 4-711
64 SDSRV Catalog Insert Collection Metadata 4-713
65 SDSRV Catalog Insertion of Metadata 4-715
66 SDSRV Catalog Search 4-717
67 SDSRV Catalog Updating Metadata 4-719
68 SDSRV Changing A Request Priority 4-721
69 SDSRV Client Browsing 4-723
70 SDSRV Client Connecting to a Data Server 4-725
71 SDSRV Client Request Submission 4-727
72 SDSRV Client Searching 4-729
73 SDSRV Deleting a Queued Request 4-731
74 SDSRV Inserting Composite ESDT 4-733
75 SDSRV Inserting New ESDT 4-735
76 SDSRV Inserting Single ESDT 4-737
77 SDSRV Instantiating an ESDT 4-739

78 SDSRV Multi State Granule 4-741
79 SDSRV Op View Queued Requests 4-743
80 SDSRV Server Handling A Browse Request 4-745
81 SDSRV Server Handling a Search Request 4-747
82 SDSRV Server Request Handling 4-749
83 SDSRV Startup of a Science Data Server 4-751
84 SDSRV Subsetting and ESDT 4-754
85 SDSRV Suspend and Resume 1 Suspend Initial Closing 4-757
86 SDSRV Suspend and Resume 2 Suspend Client Side 4-759
87 SDSRV Suspend and Resume 3 Suspend Server Side 4-761
88 SDSRV Suspend and Resume 4 Resume Session Client Side 4-763
89 SDSRV Suspend and Resume 5 Resuming Session Server Side 4-765
90 SDSRV Suspend and Resume 6 Reconstructing A Ge Suspended Object 4-767
91 SDSRV Suspend and Resume 7 Restoring Server Side Session 4-769
92 SDSRV Suspend and Resume 8 Restoring Client Side Session State 4-771
93 SDSRV Update Server Configuration 4-773
94 SDSRV Validating Metadata 4-775
5.2-1 DDSRV Client Interfaces 5-2
96 ct Class Main 5-8
97 do Class Main 5-23
98 es Class Main 5-38
99 gw Class Main 5-54
100 se Class Main 5-67
101 sv Class Main 5-69
102 DDSRV Document Insert 5-74
103 DDSRV search 5-76
104 DDSRV Acquire Dynamic Model 5-78
105 DDSRV Fault Scenario Dynamic Model 5-80
10.2-1 Data Repository HWCI Block Diagram 10-5
12.2-1 Document Data Server HWCI Block Diagram 12-4

List of Tables

3.1-1 Subsystem Interfaces 3-2
3.1-2 Physical Architecture 3-7
3.2.2.7-1 Data Server Subsystem System Management GUI 3-16
4.2-1 SDSRV CI Interfaces 4-3
4.5-1 SDSRV's Components 4-776
4.6-1 SDSRV Error Categories 4-794
4.6-2 Science Data Server Subsystem Administration Management GUI 4-796

4.6.3-1 Standard Science Data Server Reports 4-797
5.5.1 DDSRV Components 5-81
5.6-1 DDSRV Error Categories 5-86
5.6-2 Document Data Server Management GUI 5-87
5.6.3-1 Standard Document Data Server Reports 5-88
10.2.2-1 Data Repository HWCI Component Descriptions 10-7
12.2-1 Access Control & Management HWCI Component Descriptions 12-4
A-1 DSS Release B Requirements Trace A-1

1. Introduction

1.1 Identification

This technical paper provides change pages for the Release B SDPS Data Server Subsystem Design 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.

1.2 Scope

The Release B Data Server Subsystem Design Specification defines the design of the subsystem. It defines the Data Server Subsystem computer software and hardware architectural design, in accordance with the ECS Level 4 requirements.

These change pages provide updated detail design material for the Science Data Server (SDSRV) and Document Data Server (DDSRV) Computer Software Configuration Items (CSCI). In addition to the detail design sections for these two CSCI, other sections of the document have been updated as necessary to reflect changes associated with these two CSCIs.

This document reflects the February 7, 1996 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 technical paper includes introductory and reference for the technical paper and changes pages that replace sections of the Release B SDPS Data Server Subsystem Design Specification (305-CD-024-002).

The following sections pertain to this technical paper only:

- Section 1 provides information regarding the identification, scope, status, and organization of this technical paper.

The following sections replace the corresponding section of the Release B SDPS Data Server Subsystem Design Specification (305-CD-024-002):

- Section 3 provides an overview of the Subsystem, its interfaces, its structure, and the overall approach to its design.
- 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 10 describes the design of the Data Repository (DRPHW) HWCI.
- Section 12 describes the design of the Document Data Server (DDSRVHW) HWCI.
- Appendix A lists the correspondence between ECS Level 4 requirements and the components of the Data Server Subsystem.

1.4 Status and Schedule

Submittal of these change pages to DID 305/DV2 satisfies the requirements specified in the Contract Data Requirements List (CDRL) of NASA Contract NAS5-60000. This submittal contains changes and refinements made to the design of the Data Server Subsystem SDSRV and DDSRV CSCI since the SDPS Release B Incremental Design Review (IDR) and Critical Design Review (CDR) and reflects changes to the design which resulted from those reviews.

3. Subsystem Overview

3.1 Introduction and Context

The Data Server Subsystem has the responsibility for storing, searching, and retrieving earth science and related data. The subsystem provides data repositories and management capabilities necessary to safely store data on a permanent basis. The subsystem stages data needed for data processing by the Data Processing Subsystem. The subsystem organizes and stores its data by data types, and provides advanced search capabilities and processing services on those data types, in support of earth science users. The Data Server Subsystem distributes data to users either electronically or on physical media. It provides administrative capabilities necessary to operate and manage its data, hardware, and software.

Table 3.1-1 lists the Data Server Subsystem interfaces. These interfaces are summarized as follows:

- The Data Server Subsystem provides advertisements, for data types and corresponding data type services, to the Interoperability Subsystem.
- The Data Server Subsystem stores data ingested by the Ingest Subsystem, stores data produced by the Data Processing Subsystem, and stores historic data from FOS.
- The Data Server Subsystem receives service requests for data and data type services from external clients including the Client Subsystem, the Data Management Subsystem, the Planning Subsystem, and the Data Processing Subsystem.
- The Data Server Subsystem provides data requested by service requests, to the originator of the request, by means of either electronic transfer or physical media. Alternatively, the subsystem can provide references to data in the Universal Reference format described in the Overview Volume, 305-CD-020-002.
- The Data Server Subsystem issues production requests to the Planning Subsystem as a result of data requests issued by Data Server clients. It also supports the Planning Subsystem by storing data availability schedules.
- The Data Server Subsystem provides subsystem events, status, and management information to the System Management Subsystem
- The Data Server Subsystem receives management directives from the System Management Subsystem.
- The Data Server Subsystem receives logs, policy and procedure documents, and other data from the System Management Subsystem.
- The Data Server Subsystem interfaces with CSS provided security services to authorized clients. This interface is provided via infrastructure components.

Table 3.1-2 represents the Physical Architecture of the Data Server Subsystem. The Physical Architecture is defined to be the mapping of the subsystem's applications to the hardware components in which they will execute.

Table 3.1-1. Subsystem Interfaces (1 of 6)

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

Table 3.1-1. Subsystem Interfaces (2 of 6)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
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	metadata	medium	in response to staging request
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

Table 3.1-1. Subsystem Interfaces (3 of 6)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
9	TRMM (TSDIS)	Data Server	Subscription s	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
*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	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

Table 3.1-1. Subsystem Interfaces (4 of 6)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
*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
*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
11*	Client	Data Server	Session Mgmt Requests	low	as requested
11*	Client	Data Server	Subscription s	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

Table 3.1-1. Subsystem Interfaces (5 of 6)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
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
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

Table 3.1-1. Subsystem Interfaces (6 of 6)

Flow No.	Source	Destination	Data Types	Data Volume	Frequency
*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	Status Info	low	in response to request
*21	FOS	Data Server	Acquisition Plan Schedule	medium	as required

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 new to Release B.

Table 3.1-2. Physical Architecture (1 of 2)

Software Executable	COTS/Developed	HWCI
NetworkResourceManager	Developed	DIPHW
PullMonitor	Developed	ACMHW
ScienceDataServer	Developed	ACMHW
SubscriptionServer	Developed	ACMHW
StagingMonitor	Developed	DRPHW
StagingDiskResourceManager	Developed	DRPHW
AMASS	COTS	DRPHW
Illustra	COTS	DRPHW
DistributionServer	Developed	DIPHW
ArchiveResourceManager	Developed	DRPHW
TapeResourceManager	Developed	DIPHW

Table 3.1-2. Physical Architecture (2 of 2)

Software Executable	COTS/Developed	HWCI
FAXResource Manager	Developed	DIPHW
CDROMResourceManager	Developed	DIPHW
PrinterResourceManager	Developed	DIPHW
Web Server - Netscape Enterprise	COTS	DDSRVHW
HTTP Gateway	Developed	DDSRVHW
Document Data Server	Developed	DDSRVHW
Text Search Engine - TOPIC	COTS	DDSRVHW

3.2 Subsystem Overview

3.2.1 Subsystem Structure

The Data Server subsystem is composed of the following software configuration items (CSCIs):

- The Science Data Server CSCI (SDSRV) is a software component responsible for managing and providing user access to collections of non-document earth science data.
- The Document Data Server CSCI (DDSRV) is a software component responsible for managing electronic documents.
- The Storage Management CSCI (STMGT) is a software component that stores, manages, and retrieves data files on behalf of other SDPS components. By centrally managing all DSS storage pools and devices, it provides an interface that protects STMGT users from changes in technology.
- The Data Distribution CSCI (DDIST) is a software component that is responsible for formatting and distributing data to users.

The Data Server subsystem is composed of the following hardware configuration items (HWCIs):

- The Access Control and Management HWCI (ACMHW) is a hardware component that supports the software components of the Ingest and Data Server Subsystems which interact directly with users. This HWCI also provides a level of security, isolating other hardware components from external software access.
- The Working Storage HWCI (WKSHW) is a hardware component that provides high-performance storage for the purpose of caching large volumes of data on a temporary basis.
- The Data Repository HWCI (DRPHW) is a hardware component that provides high-capacity storage for the long-term storage of data files and disk repositories for the long-term storage of granule and document metadata.

- The Distribution and Ingest Peripherals HWCI (DIPHW) provides the hardware necessary to support ingest and distribution via physical media.
- The Document Data Server Hardware HWCI (DDSRVHW) provides the hardware necessary to support the software components which provide service for the search and access of documents and their metadata.

3.2.1.1 Science Data Server CSCI

The Science Data Server CSCI is responsible for managing collections of earth science and related data, and for servicing requests for the storage, search, retrieval, and manipulation of data within those collections. The science data server performs the following functions:

- manages earth science data as logical collections of related data, using interfaces that are independent of any data formats and hardware configurations dictated by underlying storage technologies;
- manages interactive sessions with external clients;
- manages the processing of service requests from users, thereby providing a variety of services on earth science and related data;
- issues requests to the STMGT and DDIST CSCIs to perform storage and distribution services in support of the processing of service requests;
- provides advertisements to the Interoperability Subsystem, in order to announce the availability of data and services to external clients;
- manages the processing of service requests from the Ingest and Data Processing Subsystems in order to "insert" data for long-term storage and access;
- manages the processing of all service requests according to the requested priority, subject to DAAC-administered priority policies and the user profiles of the requestors; and
- manages the processing of service requests from the Data Processing Subsystem, in order to provide data to be used as input for data processing.

3.2.1.2 Document Data Server CSCI

The Document Data Server CSCI manages electronic documents and associated metadata. It supports the browsing and searching of document data types via the World-Wide-Web. The document data server performs the following functions:

- manages interactive sessions with clients;
- manages the processing of service requests from users, which includes searching for documents by keyword or text index, and requesting their distribution via an HTTP interface
- provides advertisements to the Interoperability Subsystem, in order to announce the availability of documents and related services to clients; and
- manages the processing of service requests from the Ingest Subsystem and in order to "insert" documents for storage.

3.2.1.3 Storage Management CSCI

The Storage Management CSCI is a software component that provides management services for file systems, storage devices, and media peripherals, in support of the Data Server, Ingest, and Data Processing Subsystems: The Storage Management CSCI provides the following capabilities:

1) It manages high-capacity storage hardware of the DRPHW HWCI and provides associated file access and file management services, using a combination of COTS and custom software. The STMGT CSCI provides a consistent file storage and retrieval interface between the SDSRV CSCI and the file management software, independent of the underlying file management and storage technologies.
2) It manages the high-performance storage hardware of the WKSHW HWCI and provides associated file access and file management services, using a combination of COTS and custom software. The resulting storage facility supported by this combination of software and hardware is called "working storage".
3) It manages a user-accessible "pull" area used to stage data for distribution via FTP.
4) It manages the peripherals used to distribute data to physical media.
5) It manages the storage hardware that composes part of the ICLHW CSCI in the Ingest Subsystem.

The STMGT CSCI supports a data referencing capability based on the use of unique file names which do not change even if the location of the data changes within the resources managed by the CSCI. STMGT clients can use these identifiers to reference the data at any time, but the STMGT has the flexibility to move the data as needed according to operational or administrative necessities.

The STMGT CSCI supports the implementation of changeable policies which govern the establishment of operational priorities. It supports the allocation of storage resources based on user priority.

3.2.1.4 Data Distribution CSCI

The Data Distribution CSCI is responsible for formatting and distributing data to users at the direction of the SDSRV CSCI and the DDSRV CSCI. Data is distributed either electronically or by means of physical media. The DDIST CSCI performs the following functions:

- Accepts requests, from the SDSRV CSCI and the DDSRV CSCI, to distribute data which has been staged to either working storage (WKSHW HWCI), or within the DRPHW HWCI;
- Directs the STMGT CSCI to transfer data to the pull area, for user-initiated electronic distribution;
- Directs the STMGT CSCI to transfer data to users electronically, either via FTP or via FAX.
- Transfers data managed by the STMGT CSCI to physical media, including 8mm tape, 4mm tape, 6250 bpi 9-track tape, 3480/3490 tape, or CD-ROM.
- Supports operator management of distribution requests by allowing operators to view, cancel, suspend/resume, and change the priorities of requests.

3.2.1.5 Access Management HWCI

The Access Control and Management HWCI consists of hardware that supports the software components of the Ingest and Data Server subsystem which interact directly with users. The ACMHW HWCI is connected to other ECS hardware components through a secure interface, ensuring that unauthorized network users cannot access those components. The ACMHW HWCI hosts the following software:

- the SDSRV CSCI
- the DDSRV CSCI
- the INGST CSCI (Ingest Subsystem) software required to manage the ingest of non-Level 0 data by electronic means
- the STMGT CSCI software required to manage the user-accessible "pull" area used for data distribution

3.2.1.6 Working Storage HWCI

The Working Storage HWCI is a hardware component that provides high-performance storage for the purpose of caching large volumes of data on a temporary basis. The storage facilities of the WKSHW HWCI are used by:

- Data Processing Subsystem for the caching of intermediate data products used as input for subsequent processing;
- SDSRV CSCI to stage data for processing by the Data Processing Subsystem;
- INGST CSCI to prepare L0 data prior to "insertion" by the SDSRV CSCI.
- STMGT CSCI to stage data until it can be stored in the long-term storage facilities of the DRPHW CSCI;
- SDSRV to temporarily store data for post-retrieval processing, prior to distribution.

Access to the WKSHW is provided only by STMGT CSCI interfaces.

3.2.1.7 Data Repository HWCI

The Data Repository HWCI provides high-capacity storage hardware for the long-term storage and maintenance of data files as well as hardware and storage for persistent metadata. This hardware includes mass-storage devices, staging disks, DBMS repository disks and associated processors. This component is designed to support the expansion of storage capacity, and the inclusion of multiple storage technologies as appropriate to the requirements of each site. The Data Repository HWCI is managed by the STMGT CSCI for the storage and maintenance of data files and the SDSRV CSCI for the storage and maintenance of persistent metadata.

3.2.1.8 Distribution and Ingest Peripherals HWCI

The Distribution and Ingest Peripherals HWCI provides the peripherals and associated processors required for (1) the ingest of data from physical media, and (2) the distribution of data to physical media. The DIPHW HWCI hosts the following software:

- the DDIST CSCI software used to manage the distribution of data via physical media ;
- the INGST CSCI (Ingest Subsystem) software used to manage the ingest of data via physical media;
- the STMGT CSCI software used to manage the peripherals used to read from, and write to, physical media.

3.2.1.9 Document Data Server HWCI

The Document Data Server HWCI provides the hardware to support software components which provide the services for document search and retrieval. This hardware CI hosts the COTS Web Server (Netscape Enterprise), the COTS Text Search Engine (TOPIC by Verity, Inc.), and the custom developed DDSRV server and the HTTP gateway.

3.2.2. Subsystem Design Rationale

3.2.2.1 Support for Earth Science Research

The Data Server Subsystem is required to support earth science research by providing information-rich logical data collections and providing efficient search, browse, reduction, transformation and access methods on the data.

The Data Server Subsystem organizes each data collection as a set of logical data types. Each data type has an associated set of storage, retrieval, and processing services that are available to the science user on request. From the perspective of the science user, these data type services are provided using methods and interfaces that are independent of any data formats and hardware configurations dictated by underlying storage technologies.

The logical data types and their associated data type services are implemented with Earth Science Data Type classes (ESDTs). ESDTs are implemented by combining off the shelf DBMS technology with software developed to support data characteristics and operations (e.g. coordinate transformations) that are unique to the data type and are unsupported by the DBMS.

3.2.2.2 Evolvability

The Data Server Subsystem is required to be adaptable to changing technologies and new requirements. As a result, the subsystem design accommodates the widespread use of COTS software products, custom encapsulation software, and configuration-controlled APIs..

3.2.2.3 COTS Encapsulation

The Data Server Subsystem employs COTS database management systems (DBMS) and COTS file storage management systems (FSMS) to provide much of the subsystem's overall functionality. These COTS products are encapsulated with "wrapper classes", as shown in Figure 3.2.2-1. The wrapper classes translate between the vendor-specific interfaces provided by the COTS products, and the configuration-controlled interfaces of the Data Server Subsystem. The replacement of a

given COTS product requires the modification of the wrapper class, only. Encapsulation of COTS products through wrappers simplifies the replacement of those products, and supports the use of combinations of different products.

3.2.2.4 DBMS Technology Selection

ECS has driving requirements to handle complex data types and queries, with a strong emphasis on spatial queries. Spatial queries involve complex operations on spatial data types. The notion of the Earth Science Data Type is an object-oriented concept which encapsulates data structure with the operations that are useful to perform on data having that structure.

A DBMS is required which can combine the management of ECS data structures with the performance of the complex operations required to search and access those structures. Relational database management products, though mature and widely used, are not designed to meet these requirements. The Illustra Object-Relational DBMS product has been selected to implement the storage of SDSRV metadata and documents in the Release B Data Server Subsystem. Illustra provides a "server engine" which supports complex operations on complex data types, including some operations

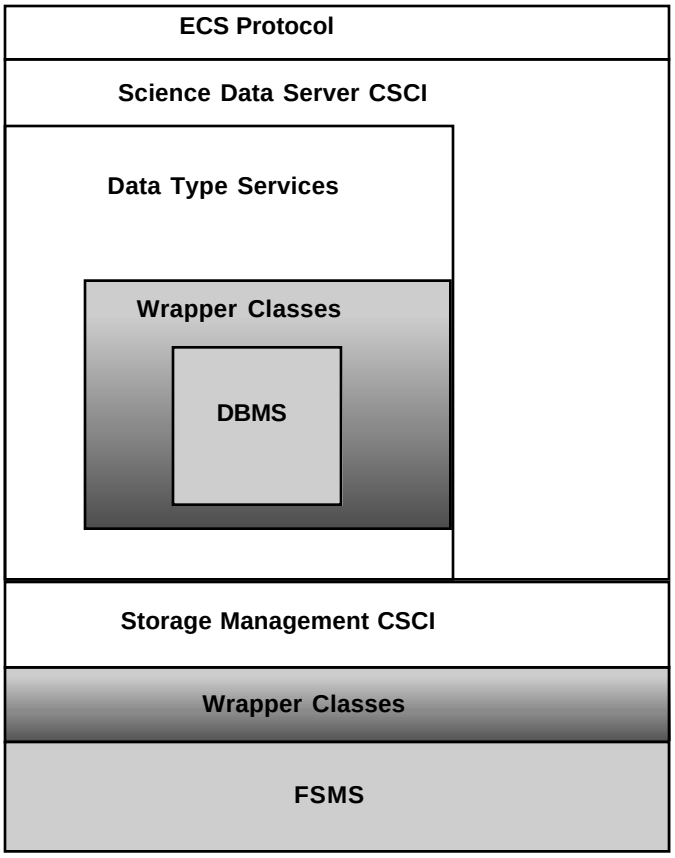


Figure 3.2.2-1. Encapsulation of COTS Products in the Data Server Subsystem

which are particularly well suited to spatial queries. In addition, the product supports the integration of user-defined operations into the server engine, along with the product's intrinsic operations.

Use of Illustra in Release B requires revision of some Release A software, such as the Sybase Data Definition Language (DDL) for the SDSRV CSCI database schema and the underlying Sybase stored procedures. The database wrappers are changed in order to support the Illustra interfaces. Most of the database load software related to the initial load of the SDSRV for Release B as well as for data migration purposes is also revised to implement the Illustra database.

3.2.2.5 The Role of FSMS Products in ECS

The ECS program will make use of COTS file storage management systems to access and manage the storage of ECS data files held in mass-storage devices.

Most FSMS products rely on a hierarchical storage approach ("hot" data on disk, "cold" data on tape) where large volumes of data are stored in some mass-storage facility (e.g. optical disk, tape library), or a disk-based caching mechanism is used to compensate for the relatively slow performance of the mass-storage facility. Hierarchical storage is used in the Data Server Subsystem for managing the user pull area. However, the total volume of data managed by ECS is large enough that an FSMS product that utilizes a disk-based caching mechanism is used to manage working storage and the permanent repository.

The Data Server subsystem employs a cost-effective cache management scheme in working storage and the permanent repository that augments the generic COTS FSMS caching mechanisms employed by FSMS products. The scheme is supported by the following design features:

- the separation of working storage from the staging disks managed by the FSMS products
- the use of working storage for caching data generated and used by the Data Processing Subsystem
- the separation of the data distribution "pull area" from working storage

3.2.2.6 FSMS Technology Selection

FSMS products can generally be divided into three basic groups according to the method used to implement their file system interfaces:

1) UNIX Resident Products - These products (e.g. EMASS FileServ, EMC EpochServ) are integrated with the Unix File System so that all native UNIX file access methods can be used to access data resident on the devices managed by the FSMS.
2) Separate File System Products - These products (e.g. UniTree products) implement proprietary file systems and access methods which are independent of the UNIX file system.
3) Hybrids - These products (e.g. EMASS AMASS, Qstar Mastermind, ANT Metior, LSC's Sam-FSuse) implement file management systems that are separate from UNIX but, like NFS, are linked to the UNIX file system via the Virtual File System (VFS) Virtual Node (vnode) interface provided by the Unix kernel. As a result, these products allow the use of native access methods but do not manage files using inodes.

The FSMS product selected for Release A was the Archival Management and Storage System (AMASS) product marketed by EMASS Inc. The Release B design is also based on this product. The features of the AMASS product are summarized as follows:

- The VFS-linked separate file system design allows all Unix File System access methods to be employed (e.g. ftp, rcp, uucp, nfs, RPC, native, etc.).
- The AMASS product is not subject to limitations imposed by Unix 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.
- AMASS organizes files as groups of blocks which can be individually retrieved. By contrast, other FSMS products require that the entire contents of the file be staged even only a portion is required for access.
- AMASS utilizes a disk based I/O buffer for communications rate matching between disk and tape resources. The I/O buffer scheme supports the encapsulation of storage and data viability functions while allowing the Data Server Subsystem to manage the associated Working Storage.

3.2.2.7 Layering of GUIs for System Operations

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 Table 3.2.2.7-1. Each of the CI GUI interfaces are described in the appropriate CI section.

Table 3.2.2.7-1. Data Server Subsystem System Management GUI (1 of 2)

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 3.2.2.7-1. Data Server Subsystem System Management GUI (2 of 2)

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

3.2.3. Data Server Subsystem Common Design

This section describes some DSS design components that are common across DSS CSCIs.

3.2.3.1. Costing/Utilization

While it is the responsibility of the Billing and Accounting Subsystem (BAAS) to provide a mechanism of pricing services and to the client account billing, the DSS components are responsible for providing resource utilization. It is the responsibility of the BAAS to convert that resource utilization into pricing information. In this light, we look at the DSS as responsible for providing "costing" (in terms of resource utilization) information, and BAAS for converting those costs into pricing tables.

Since each DSS component has a responsibility to provide costing information, a single design mechanism has been defined that will support a DAAC-configurable resource costing, on a service-by-service basis. A set of classes that are to be re-used across the DSS CSCI for utilization have been designed. For purposes of this document, these classes are documented as part of the storage management CSCI, in the Utilization (Uz) CSC. All of the classes have a prefix of DsUz.

This model is based on the concept of determining the resources that are utilized to fulfill a given service request, then determining how much they are/would be utilized in fulfilling that request. There is a table that contains rows of application, service and resource tuples. The application is included in order to allow support of multiple DSS applications from the same table. Each service may require multiple resources used. In this case there would be multiple rows in the table for that service, one for each resource used. This table is to be implemented using DBMS technology, which will allow DAACs to modify the resources used on a service-by-service (by-application) basis.

Once all the resources for a service are collected, each will support its own costing based on the amount of data it is being applied to. The resource will convert the volume of data, in megabytes, into appropriate units, based on a static utilization factor. This allows different types of resources to vary in their "cost" per volume of data.

The detailed design of these classes is presented within the Storage Management section of this document.

3.2.3.2. Generic Client Framework

Early in the development cycle of Release A, the DSS development team recognized the value of a generic framework to support development of distributed objects in a client/server architecture. The goals of this framework were to support distributed objects for a variety of client/server applications, isolate the generic mechanisms from the application specific logic, allow extension on both the client and server sides of the application, support both state-less and stateful connections and abstract away all the underlying distributed object technology (currently OODCE), allowing for a technology migration with no impact to client software interfaces.

The result of this effort is a framework, implemented as a set of class libraries, that supports not only distributed objects, but also client connections to the server. These connections are easily established and broken. The connections can be either state-less or state-ful. State-less connections support short-term single activity sessions with clients, while state-ful connections

support longer term, multiple activity sessions, with a capability of queueing requests. The types of connections are built to support application specific behavior through specialization.

The basic information passed between application clients and their servers are commands and requests. As part of their usage of the framework, applications can specialize commands and requests, adapting them to implement application specific behavior. The requests that are submitted can be synchronous or asynchronous. Asynchronous request use a callback function to notify clients of changes in state, including completion.

This framework is used to support several of the application processes being developed as part of the DSS. This is being implemented within the Release A effort.

3.2.4. Data Server Subsystem Use of Key Design Mechanisms

This section is provided to describe how the Data Server Subsystem will utilize some of the ECS Key Mechanisms. These Key Mechanisms are being developed as a part of the normal development effort, under the responsibility of the Infrastructure Design Group and CSS for Release B. The design details of each of these Key Mechanisms can be found in the volume titled Release B CSMS Communication Subsystem Design Specification, 305-CD-028-002, of this document.

3.2.4.1. Universal References

Universal References are the Key Mechanism used to uniquely identify an entity within ECS. An application creates a specific use of the Universal Reference Key Mechanism by specializing two classes, EcUrUR and EcUrURProvider, for each type of UR that is offered by the application to the ECS.

Within the DSS, we have identified the need for 5 different types of URs that will inherit from the EcUrUR and EcUrURProvider classes. These five types of data server URs are:

GranuleUR

DsConnectionUR

SubscriptionUR

EventUR

DsServerUR

In classes derived from each of the two base classes, two virtual functions must be specialized. These virtual functions are:

EcUrUR::Externalize

EcUrUR::Internalize

EcUrURProvider::Extract

EcUrURProvider::Reconstitute

The basic implementation of URs based on this model is:

UR:<URID>:<implementation specific data><0>

The UR and colons are literals. The <URID> is defined at UR registration time, and will be defined by the CSS UR implementors. The <implementation specific data> is where application-specific attributes needed to uniquely identify those entities represented by the UR are placed.

3.2.4.1.1. GranuleUR

The GranuleUR will be used to represent a granule in the data server. Its <implementation specific data> is defined to be:

<serverUR><GranuleID>,

where <serverUR> is as defined below and <GranuleID> is defined as:

<type>:<subtype>:<uniqueID>.

<type>::16 bytes

<subtype>::16 bytes

<uniqueID>::20 bytes

The total size of a GranuleUR is 201 bytes.

3.2.4.1.2. DsConnectionUR

The DsConnectionUR will be used to represent a client's connection with the data server. This will be primarily used when a session is to be suspended and resumed. Its <implementation specific data> is defined to be:

<serverUR><ConnectionID>,

where <serverUR> is as defined below and <ConnectionID> is defined as:

Connection::<uniqueID>.

<uniqueID>::8 bytes

The total size of a DsConnectionUR is 167 bytes.

3.2.4.1.3. SubscriptionUR

The SubscriptionUR will be used to represent a specific subscription that has been submitted to a subscription server. Its <implementation specific data> is defined to be:

<serverUR><SubscriptionID>,

where <serverUR> is as defined below and <SubscriptionID> is defined as:

Subscription::<uniqueID>.

<uniqueID>::8 bytes

The total size of a SubscriptionUR is 169 bytes.

3.2.4.1.4. EventUR

The EventUR will be used to represent a specific event that has been registered with a subscription server. Its <implementation specific data> is defined to be:

<serverUR><EventID>,

where <serverUR> is as defined below and <EventID> is defined as:

SubscriptionEvent::<uniqueID>.

<uniqueID>::8 bytes

The total size of an EventUR is 174 bytes.

3.2.4.1.5. DsServerUR

The DsServerUR will be used to represent a specific executing data server application that is a service provider. Examples of these include running ScienceDataServer and SubscriptionServer applications. The <implementation specific data> of a DsServerUR is defined to be:

<CDSname>,

where <CDSname> is the uuid of the process as it was registered with the DCE Naming Service.

<serverUR>::UR:<URID>:<CDSname>

<URID>:: TDB (by CSS/DE)

<CDSname>:: 128 bytes

The total size of a DsServerUR is 147 bytes.

3.2.4.2. Distributed Object Framework

The use of the Distributed Object Framework (DOF) key mechanism within DSS is encapsulated in the generic client framework (discussed above) or in the use of Server/Request Framework. Significant effort was been taken to eliminate direct use of the DOF within DSS applications.

3.2.4.3. Event Detection/Logging

The Data Server will use MSS Event services to log a number of system level events. The types of events are best categorized as general events and CSCI based events. While the entire list of MSS events is not yet completely defined, the following is a list of those MSS events that are required by the Data Server applications in order to support the appropriate system level management. In this list we are not accounting for any fault, error or warning-type of events. Also not included in this list are any instrumentation events.

3.2.4.3.1. General Events

Process Startup

Process Normal Shutdown

3.2.4.3.2. SDSRV Events

Client Session Established

Client Session Suspended

Client Session Resumed

Client Session Terminated

Request Received

Request Queued

Request Activated
Request Completed
Request Deleted
Search Started
Search Completed
OnDemand Process Initiated
OnDemand Process Completed
Granule Insertion

3.2.4.3.3. DDIST Events

Distribution Request Received
Distribution Notice sent
Distribution Request Completed
Distribution Operator Intervention

3.2.4.3.4. STMGT Events

Resource Allocation
Resource Deallocation
Archive connection established
Archive connection terminated
Store Service complete
Retrieve Service complete
Resource not available
Resource in maintenance
Resource on-line

3.2.4.3.5. DDSRV Events

Request Received
Request Completed
Search Started
Search Completed
Document Sent
Document Inserted
Document Metadata Inserted
Document Text Index Completed

3.2.4.4. Process Framework

This Key Mechanism provides ECS application teams a common framework that facilitates the development of client/server ECS applications. In general, this Key Mechanism defines a classification of ECS applications, and a set of services provided by each classification. This classification is implemented as a set of classes. The leaf-node types in the classification tree are designed to be specialized by the application, allowing for the application to develop its own implementation for various process services. The Process Framework includes a special case, Managed Process Framework for server applications.

Each of the ECS-built applications that comprise the DSS are types of ManagedProcesses. As managed processes, each will be specializing the EcPfManagedServer class. This specialization will include implementations of the PfShutdown and PfGetShutdownSec member functions. During coding, the specialized classes will be instantiated immediately after the main () function. The following is an example of a snippet of code instantiating the type of ManagedProcess object.

```
.
.
.
main()
{
        <DsXXCSCI>ManagedProcessmyApp(argc, argv, configfile);
.
.
.
myApp.PfStart();
}
```

The details of the CSCI application specializations, as well as the logic for the implemented member functions is described in detail within each of the CSCI sections of this document.

3.2.4.5. Mode Management

Mode Management within DSS will be handled in two ways. The first is within ECS-built software, which will utilize the Managed Process Framework (MPF). Within the MPF, various environment and configuration parameters will be managed, allowing the Managed Process to be running in the specified mode. At this time there is expected to be no mode specific logic required for our ECS-build applications. Their mode will be dictated by their configuration, the environment in which they are executing, and directory services for mode-specific name resolution.

The second technique for managing mode within DSS is related to how the COTS products used are mode aware. Since COTS products are not themselves MSS managed applications the MPF does not support their configuration and execution environment. COTS products role in supporting various modes of operation will be described within the appropriate CSCI sections of this design document.

This page intentionally left blank.

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 instantiations 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 Subvolume 305-CD-22-002 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 Data Type Descriptors, one for each ESDT, as formal descriptions of the ESDT, including its attributes and operations. 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 (and made available through Data Dictionary services); 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 of the Data Management Subsystem. (See Subvolume 305-CD-23-002.). 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 B interfaces to other ECS CSCIs.

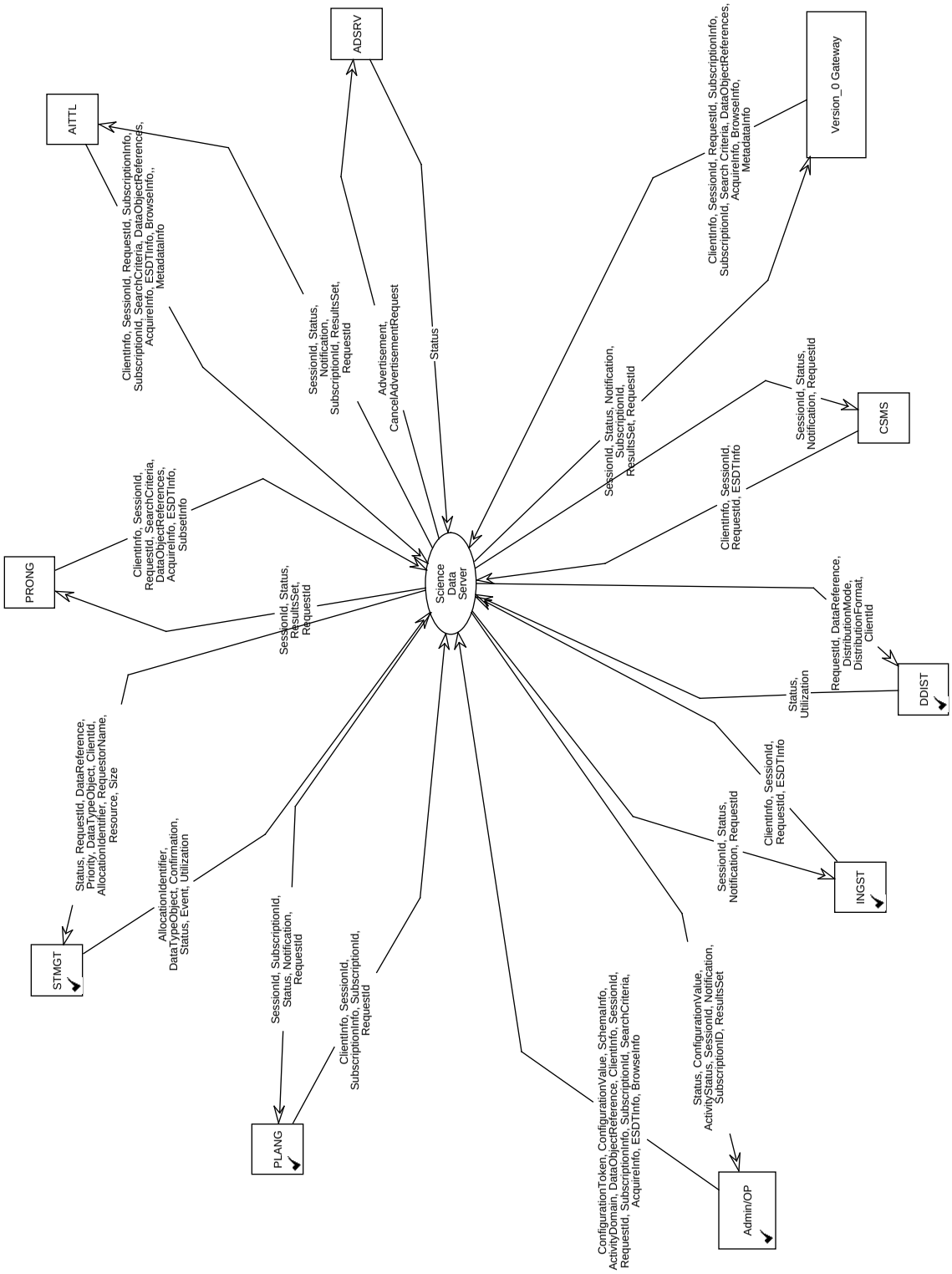


Figure 4.2-1 SDSRV CI Interfaces

Table 4.2.1. SDSRV CI Interfaces (1 of 3)

Interface	Interface Input	Interface Output	Description
ConnectServer (Session)	DsClESDTReferenceCollector::DsClESDTReferenceCollector ()	*DsClESDT ReferenceCollector	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).

Table 4.2.1. SDSRV CI Interfaces (2 of 3)

Interface	Interface Input	Interface Output	Description
Search	DsClES-DTReferenceCollector::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".
Suspend Session	DsClES-DTReferenceCollector::Suspend()	GlUR	This interface is used by SDSRV clients to suspend the current session. The SDSRV will return the UR of the suspended session.
Resume Session	DsClES-DTReferenceCollector::Resume(MSS_UserProfile, GlUR)	GlStatus	This interface is used by SDSRV clients to resume a previously suspended session. The SDSRV will return the status of the resume activity.
GetResultsSet	DsClRequest::GetResults()	GlParameterList	This interface is used by SDSRV clients to assess the results of a completed request. A ResultsSet is a implemented as a GlParameterList.
ListContents	DsClES-DTReferenceCollector::GetUR(DsClRequest)	GlParameterList	This interface is used by SDSRV clients to get a UR for each granule represented in the clients current session.

Table 4.2.1. SDSRV CI Interfaces (3 of 3)

Interface	Interface Input	Interface Output	Description
Acquire	DsClES-DTReferenceCollector::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	DsClES-DTReferenceCollector::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	DsClES-DTReference::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.

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.

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 4.3-20. Each of the objects in this figure is de-

scribed in the subsequent paragraphs. The Science Data Server Object model consists of a series of 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 class diagram. The public view is defined as those classes labelled as [PUBLIC]. Due to complexity, some class categories are modeled in more than one 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).

4.3.1 ac Class Category

4.3.1.1 Overview

Figure 1: Main

4.3.1.2 ac Classes

4.3.1.3 DsAcACRIM Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the ACRIM instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsAcACRIM();

~DsAcACRIM();
```

Operations:

4.3.2 ad Class Category

4.3.2.1 Overview

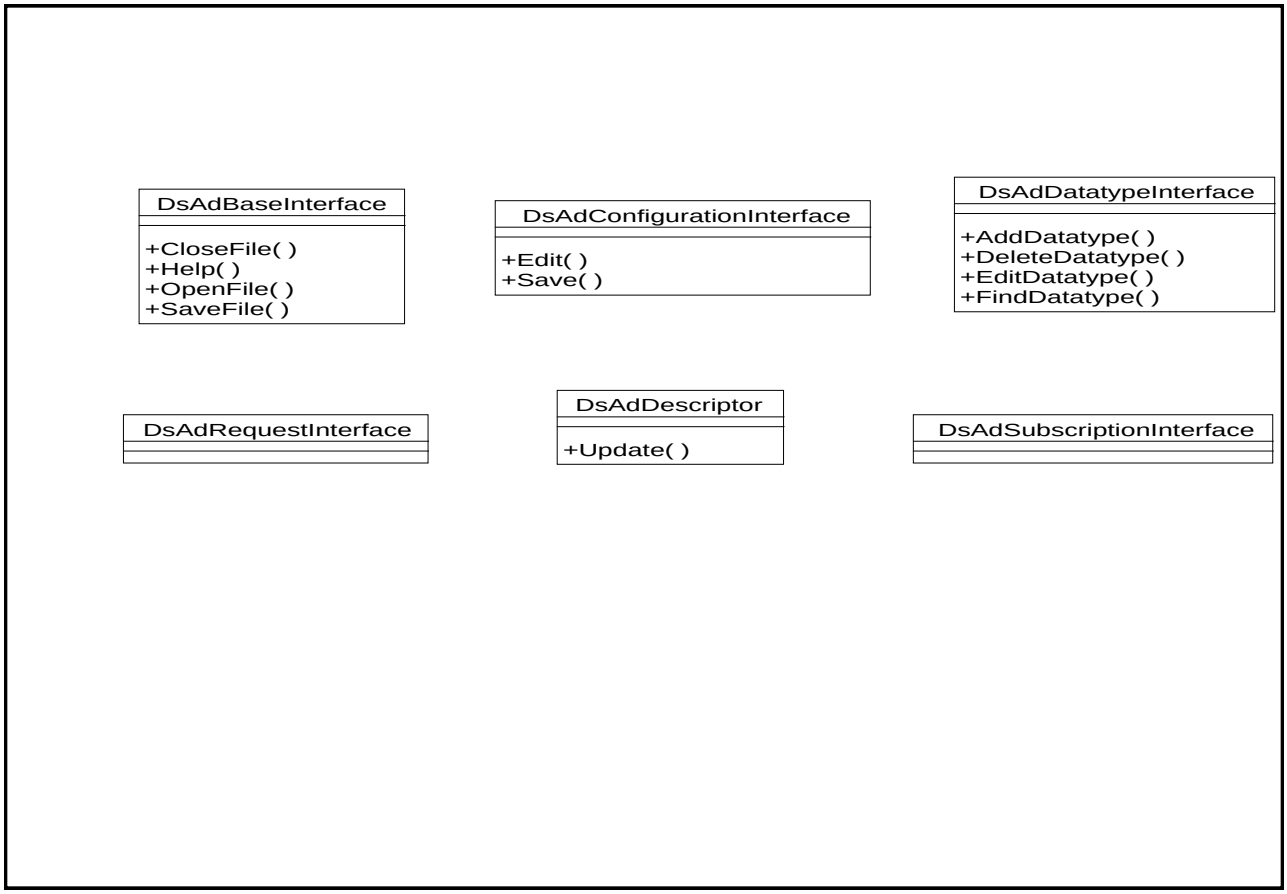


Figure 2: Main

4.3.2.2 ad Classes

4.3.2.2.1 DsAdBaseInterface Class

Overview:

Distributed Object: No This class supports the basic functioning of the operator GUIs.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsAdBaseInterface();
```

Creates an empty object.

```
~DsAdBaseInterface();
```

Removes an object of this type.

Operations:

- CloseFile

```
const GlStatus & CloseFile(fstream&);
```

This operation closes the specified file.

- Help

```
RWBoolean Help();
```

This operation provides the context-sensitive help to the operator GUI screens.

- OpenFile

```
fstream& OpenFile(RWCString filename);
```

This operation opens the specified file.

- SaveFile

```
const GlStatus & SaveFile(fstream&);
```

This operation saves the current buffer to the specified file, without closing the file.

4.3.2.2.2 DsAdConfigurationInterface Class

Overview:

Distributed Object: No This class provides the functioning to support the operator GUI screens for managing the system configuration.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsAdConfigurationInterface();
```

Creates an empty object.

```
~DsAdConfigurationInterface();
```

Removes an object of this type.

Operations:

- Edit

```
Edit(fstream&);
```

This operation allows read/write access to the provided information (which could be either configuration file or configuration database information).

- Save

```
Save(fstream&);
```

This operation saves the current buffer to disk (without closing a file).

4.3.2.2.3 DsAdDatatypeInterface Class

Overview:

Distributed Object: No This class allows operators to maintain all the datatypes in the system.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsAdDatatypeInterface(GlUR& dataserver);
```

Creates an empty object. THis constructor creates a DsAdDatatypeInterface object on the specified dataserver.

```
~DsAdDatatypeInterface();
```

Removes an object of this type.

Operations:

- AddDatatype

```
const GlStatus& AddDatatype(e, RWCString DLLfilename
DsTGeTypeID, RWCString descriptorFilename, RWCString
DLLfilename);
```

This operation allows operators to register new (approved) datatypes with the system.

- DeleteDatatype

```
DeleteDatatype(DsTGeTypeID);
```

This operation allows operators to remove old (obsolete) datatypes from the system.

- EditDatatype

```
EditDatatype(DsTGeTypeID);
```

This operation allows operators to update existing datatypes.

- FindDatatype

```
DsDeDescriptor& FindDatatype(TypeID RWCString name, RWCString
version, DsTGeTypeID);
```

This operation allows operators to find the specified datatype information.

4.3.2.2.4 DsAdDescriptor Class

Overview:

Distributed Object: No Provides more functionality than the client descriptor with the ability to update descriptors as an adminstrative interface to descriptors. Provides more functionality than the client descriptor with the ability to update descriptors as an administrative interface to them.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsClDescriptor

Attributes:

Constructors and Destructor:

```
DsAdDescriptor(RWCString& type, RWCString& vsn, GlClient&
client, GlUR& ds, RWCString& type, RWCString& vsn, GlClient&
client, GlUR& ds, iostream& strm);
```

Constructs an operations descriptor for a particular dataserver and stream.

```
~DsAdDescriptor();
```

Operations:

- Update

```
Update(GlParameterList&);
```

Update the descriptors given a list of parameters.

4.3.2.2.5 DsAdRequestInterface Class

Overview:

Distributed Object: No Provides administration/operations staff an interface to look at active and queued requests as well as status information of requests.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsAdRequestInterface(GlUR&, GlClient&);
```

Constructs the administration/operations request interface given a particular dataserver and client.

```
~DsAdRequestInterface();
```

Removes an object of this type.

Operations:

- GetClientInfo

```
MSS_UserProfile GetClientInfo(DsSrClient&);
```

This operation returns full user information for the specified client.

- GetHistory

```
istream & GetHistory(DsTSrRequestID);
```

This operation allows operators to look at all information in the system associated with the past processing of the specified request.

- GetRequests

```
DsSrRequestVector& GetRequests(DsESrPriority, DsESrState,
DsSrClient&, DsTSrProcess pid, RWCString CSCI);
```

This operation allows operators to monitor and manage requests by their current processing state (active, queued, completed). This operation allows operators to monitor and manage requests by priority. This operation allows operators to monitor and manage requests by CSCI. This operation allows operators to monitor and manage requests associated with the specified user. This operation allows operators to monitor and manage all requests in the system. This operation allows operators to monitor and manage all requests associated with a specific system process.

- GetRequest

```
DsSrRequest& GetRequest(DsTSrRequestID);
```

This operation allows operators to access full information about the specified request.

- GetResources

```
istream& GetResources(DsTSrRequestID);
```

This operation allows operators to review the resource usage of the specified request.

- SetQueueSize

```
RWBoolean SetQueueSize(int threshold);
```

Allows the operations staff to set threshold for the number of service requests to be queued for processing.

4.3.2.2.6 DsAdSubscriptionInterface Class

Overview:

Distributed Object: No This class allows operators to populate the Subscription Management screen.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsAdSubscriptionInterface();
```

Creates an empty object.

```
~DsAdSubscriptionInterface();
```

Removes an object of this type.

Operations:

- GetSubscriptionInfo

```
DsSbSubscription& GetSubscriptionInfo(GlUR& subscription);
```

Gets all the information about the specified subscription.

- GetSubscriptions

```
RWVector<DsSbSubscription> GetSubscriptions(DsESbAction action,
DsESbEvent event, MSS_UserProfile);
```

Gets a list of all the subscriptions in the system. This operation gets a list of all the subscriptions associated with the specified action. This operation gets a list of all the subscriptions associated with the specified event. This operation gets a list of all the subscriptions associated with the specified user.

- GetUserInfo

```
MSS_UserProfile GetUserInfo(DsSrClient&);
```

This operation returns all the user information about the specified client.

4.3.3 as Class Category

4.3.3.1 Overview

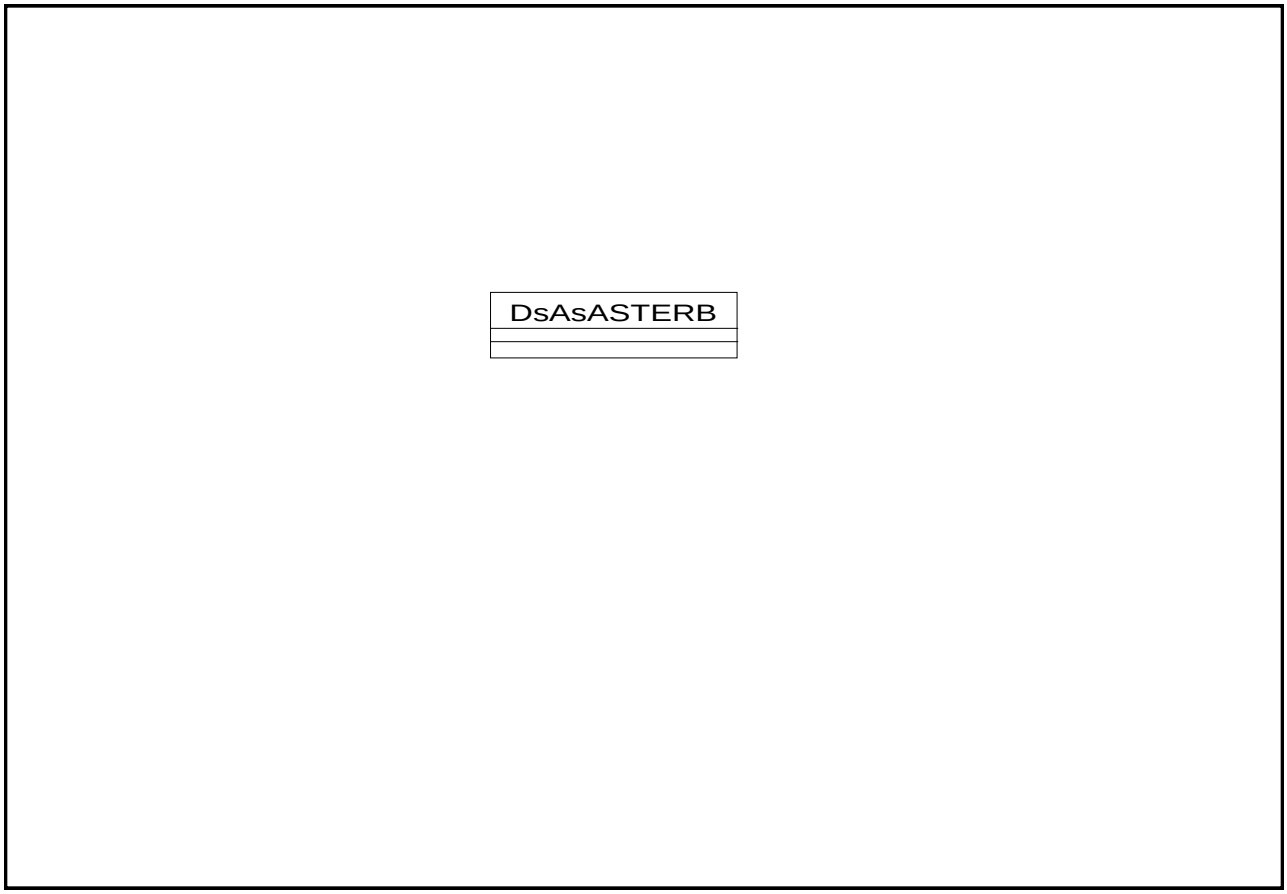


Figure 3: Main

4.3.3.2 as Classes

4.3.3.2.1 DsAsASTERB Class

Overview:

Distributed Object: No This class represents products generated from the ASTER Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsAsASTERB();

~DsAsASTERB();
```

Operations:

4.3.4 av Class Category

4.3.4.1 Overview

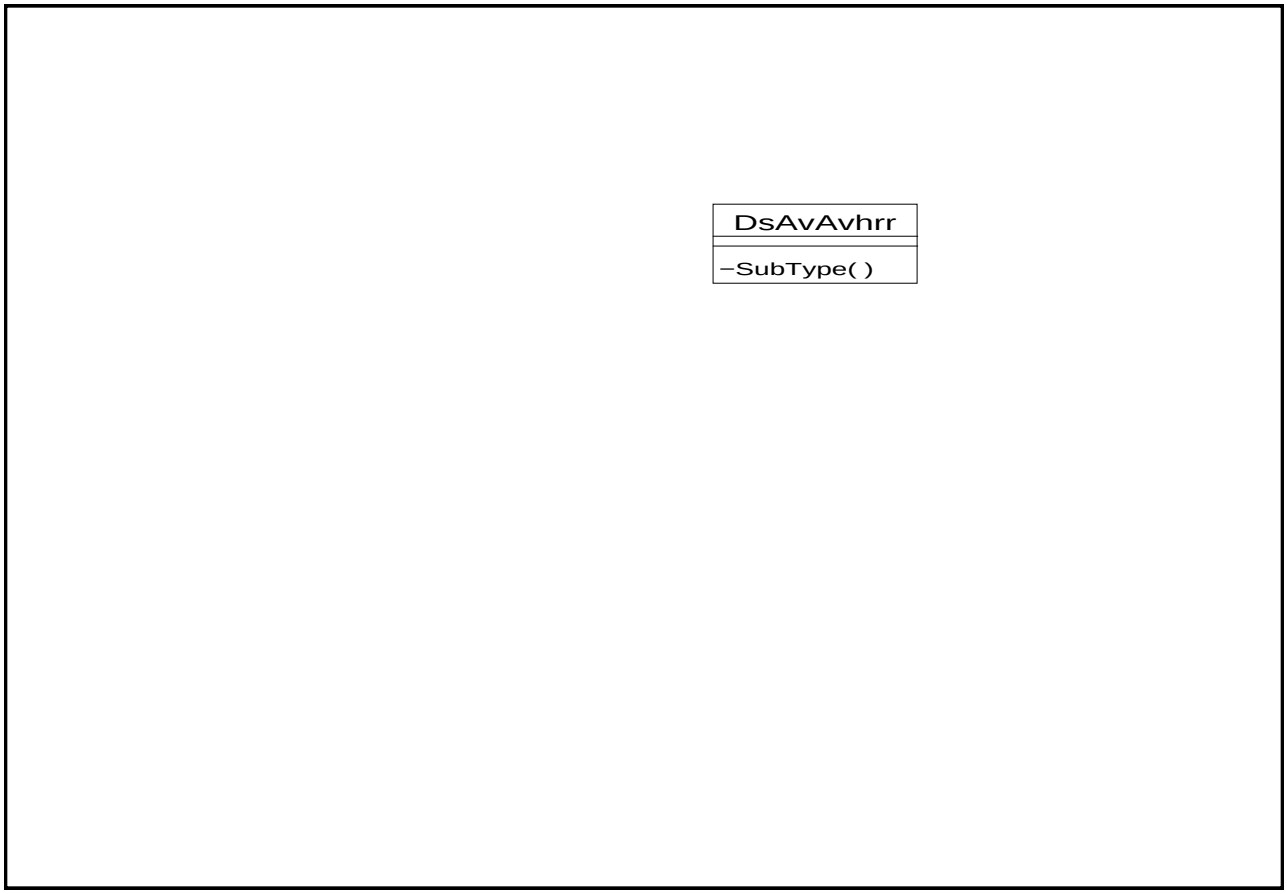


Figure 4: Main

4.3.4.2 av Classes

4.3.4.2.1 DsAvAvhrr Class

Overview:

This file becomes a dynamic library that implements the AVHRR temporary data type. The "standard" entry point DsGeNewESDT is provided by all libraries.

Distributed Object: No This class is an abstraction of the products derived from the Avhrr instrument data.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsAvAvhrr();
```

```
~DsAvAvhrr();
```

default constructor BEGIN_PROLOG

NAME ~DsAvAvhrr

DESCRIPTION Destructor for this class

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

NOTES

END_PROLOG

Operations:

- SubType

```
RWCString SubType() const;
```

BEGIN_PROLOG

NAME SubType

DESCRIPTION This function returns the datatype name of this ESDT instance.

OUTPUTS None

RETURNS An RWCString with the name of this ESDT instance.

GLOBALS None

FILES None

NOTES

END_PROLOG

4.3.5 ba Class Category

4.3.5.1 Overview

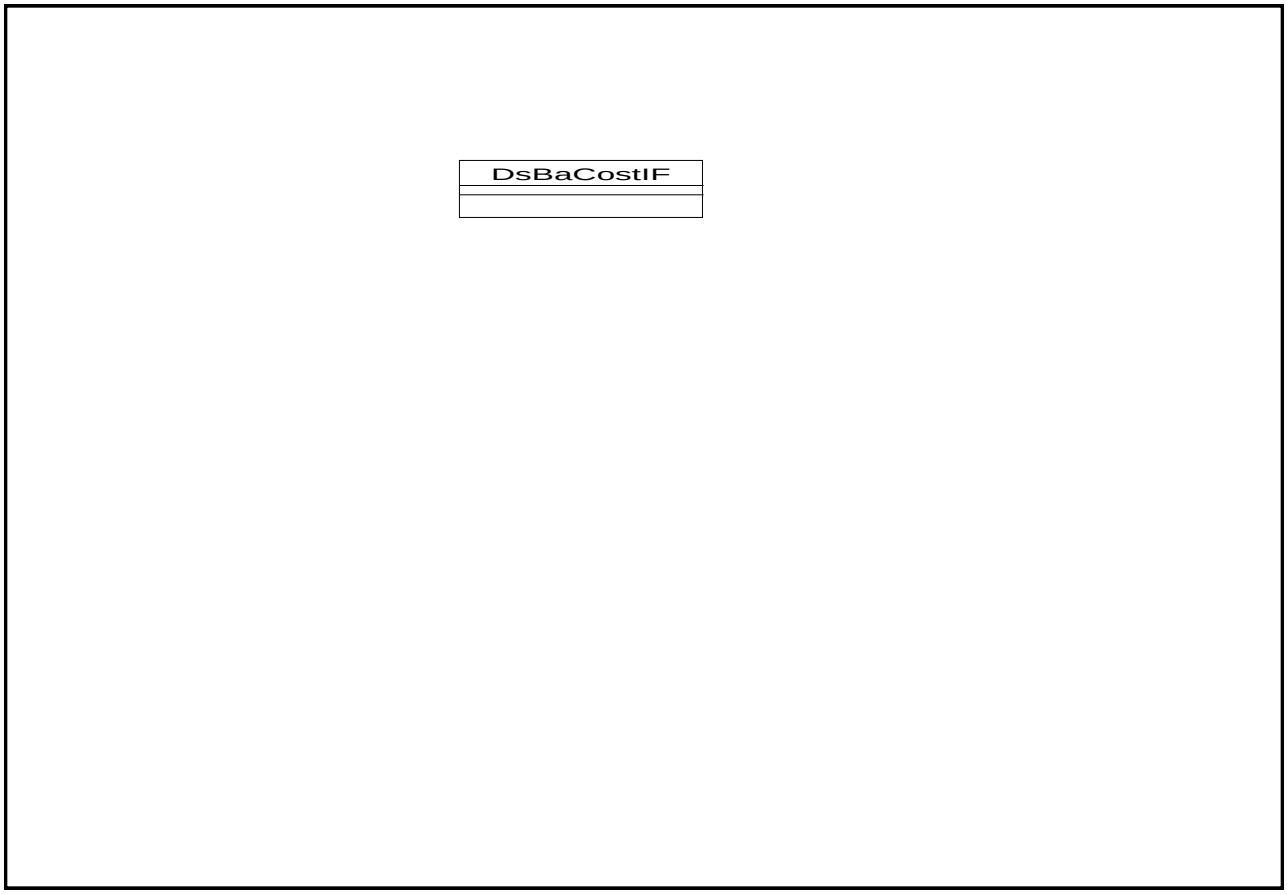


Figure 5: Main

4.3.5.2 ba Classes

4.3.5.2.1 DsBaCostIF Class

Overview:

Distributed Ojbect: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.6 ce Class Category

4.3.6.1 Overview

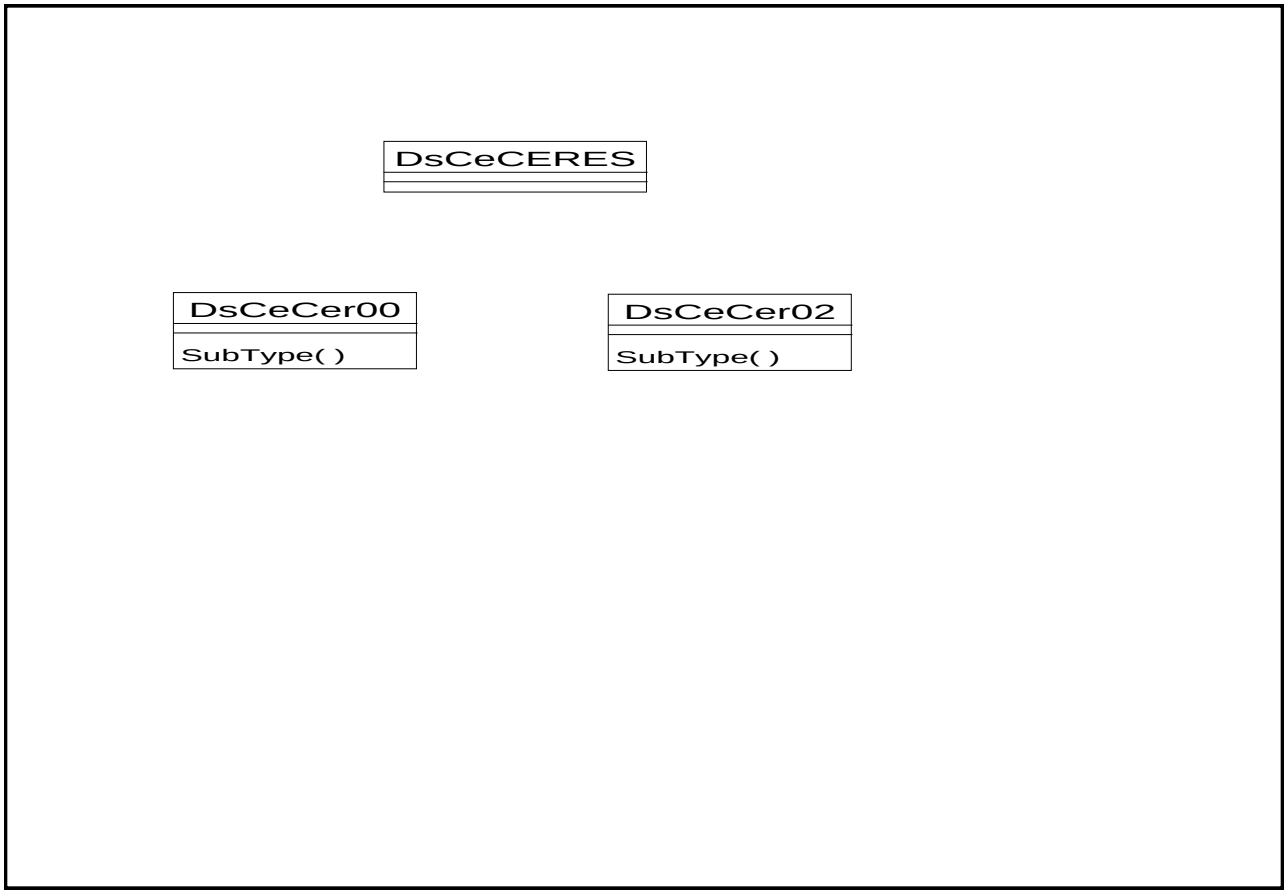


Figure 6: Main

4.3.6.2 ce Classes

4.3.6.2.1 DsCeCer00 Class

Overview:

This file becomes a dynamic library that implements the CER00 data type. The "standard" entry point DsGeNewESDT is provided by all libraries.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsCeCer00();
```

```
~DsCeCer00();
```

default constructor BEGIN_PROLOG

NAME ~DsCeCer00

DESCRIPTION Destructor for this class

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

NOTES None

END_PROLOG

Operations:

- SubType

```
RWCString SubType() const;
```

BEGIN_PROLOG

NAME SubType

DESCRIPTION This function returns the datatype name of this ESDT instance.

OUTPUTS None

RETURNS An RWCString with the name of this ESDT instance.

GLOBALS None

FILES None

NOTES None

END_PROLOG

4.3.6.2.2 DsCeCer02 Class

Overview:

This file becomes a dynamic library that implements the CER02 data type. The "standard" entry point DsGeNewESDT is provided by all libraries.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsCeCer02();
```

```
~DsCeCer02();
```

default constructor BEGIN_PROLOG

NAME ~DsCeCer02

DESCRIPTION Destructor for this class

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

NOTES None

END_PROLOG

Operations:

- SubType

```
RWCString SubType() const;
```

BEGIN_PROLOG

NAME SubType

DESCRIPTION This function returns the datatype name of this ESDT instance.

OUTPUTS None
RETURNS An RWCString with the name of this ESDT instance.
GLOBALS None
FILES None
NOTES None
END_PROLOG

4.3.6.2.3 DsCeCERES Class

Overview:
Distributed Object: No This class represents products generated from the CERES Instrument data.

Export Control: Public

Inheritance Relationships:
Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsCeCERES();

~DsCeCERES();
```

Operations:

4.3.7 cl Class Category

4.3.7.1 Overview

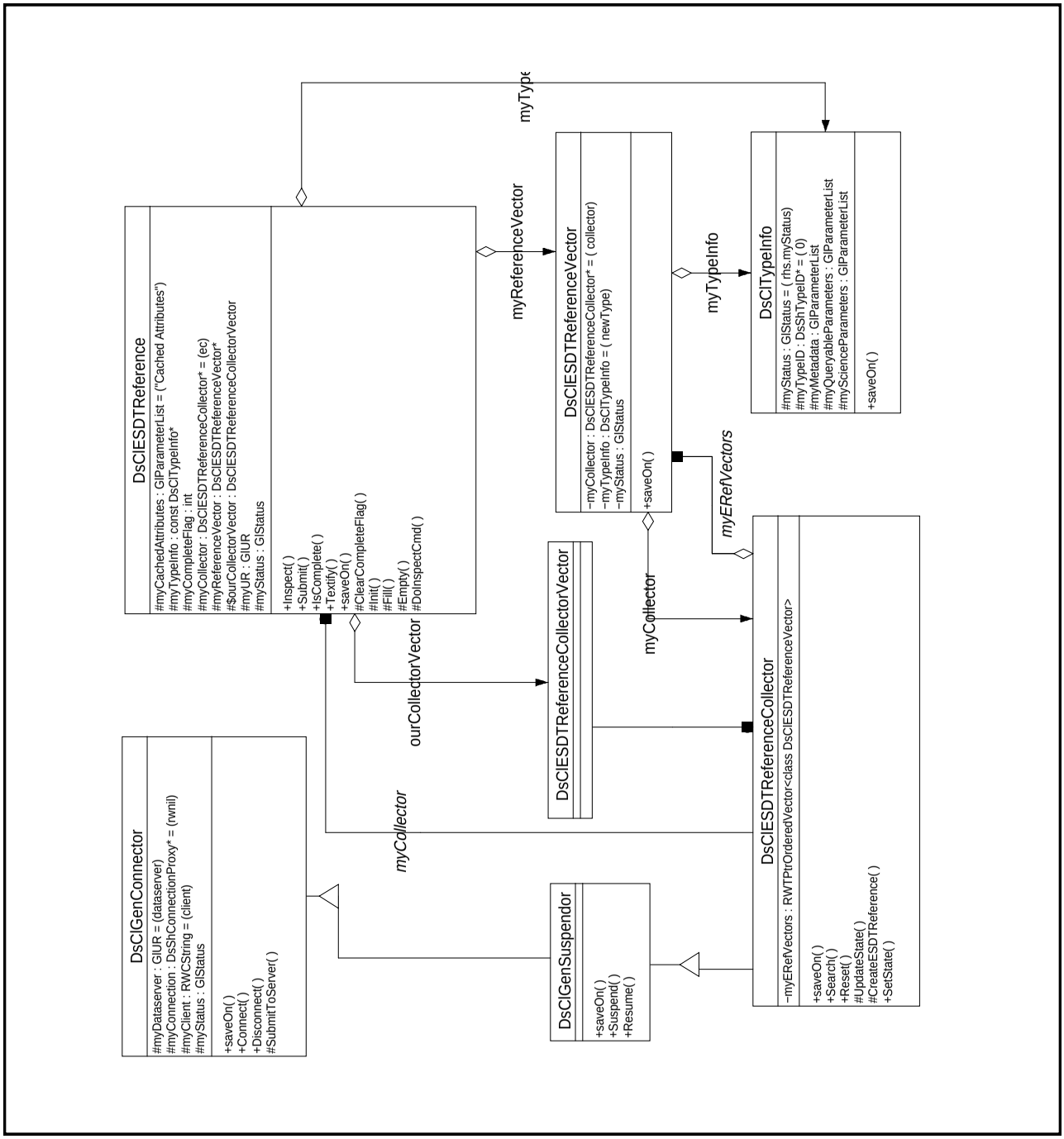


Figure 7: DsCl1

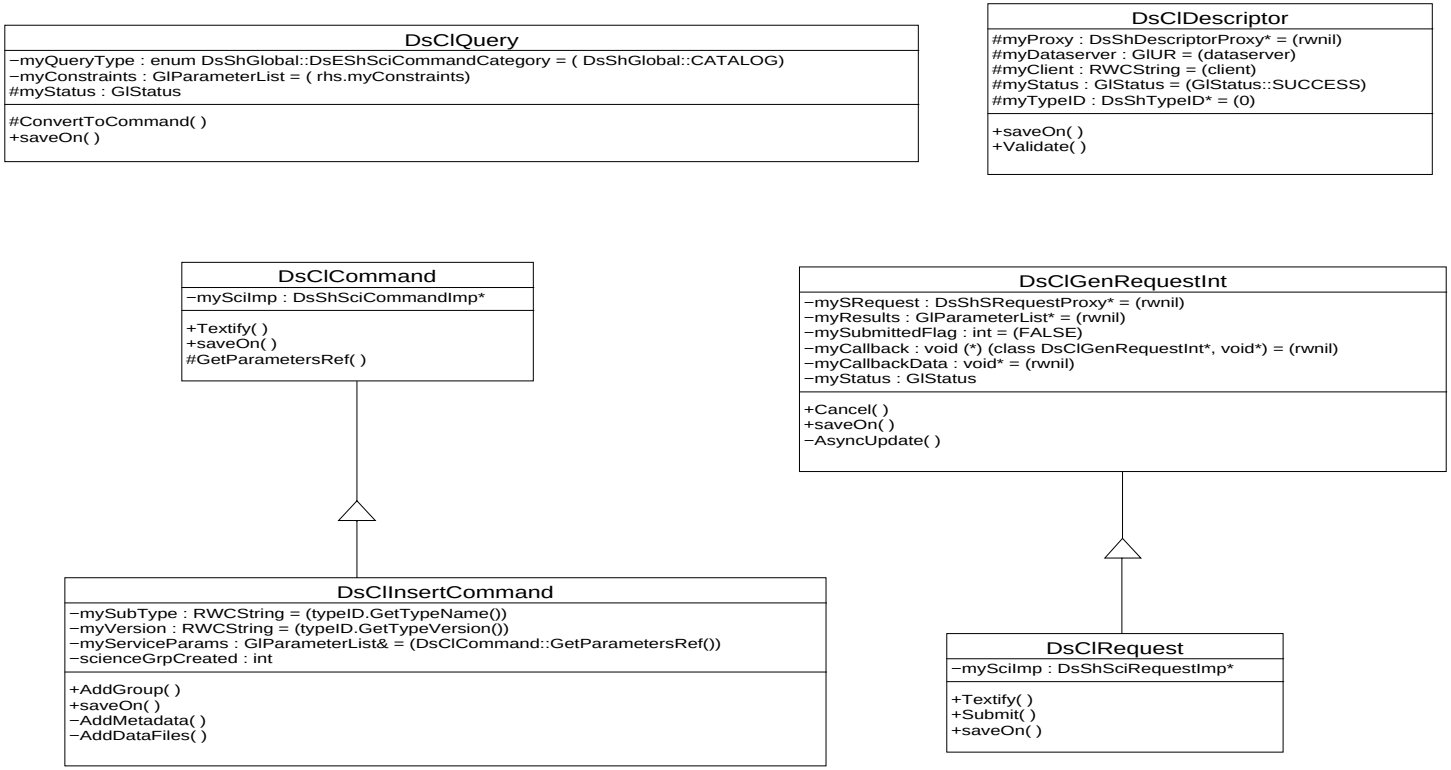


Figure 8 DsCl2

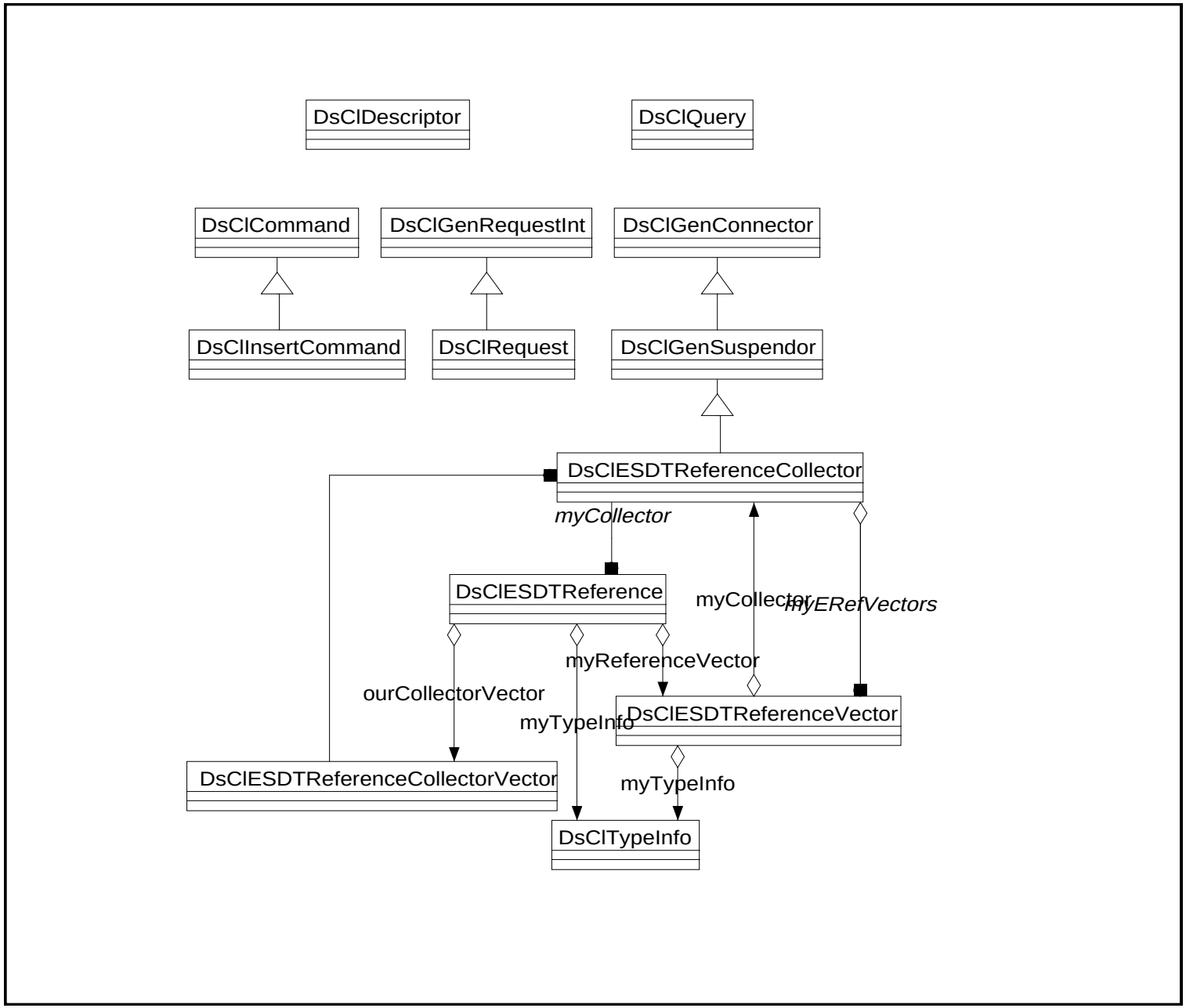


Figure 9: Main

4.3.7.2 cl Classes

4.3.7.2.1 DsClCommand Class

Overview:

Contains the definition of the DsClCommand class, used by client applications to compose commands to the data server. Provides a client view of the shared command implementation class, which is transferred to the server upon submission.

Distributed Object: No

Export Control: Public

Inheritance Relationships:

Attributes:

`mySciImp: DsShSciCommandImp*`

for access to mySciImp

Constructors and Destructor:

`DsClCommand(const DsClCommand& );`

copy constructor NOT IMPLEMENTED

`DsClCommand(const RWCString& svc, GlParameterList& pl, enum DsShGlobal::DsEShSciCommandCategory cat);`

normal constructor BEGIN_PROLOG

NAME DsClCommand

DESCRIPTION normal constructor

RETURNS nothing

END_PROLOG

`DsClCommand();`

default constructor BEGIN_PROLOG

NAME DsClCommand

DESCRIPTION default constructor

RETURNS nothing

END_PROLOG

```
virtual  ~DsClCommand();
```

category, default WC default destructor BEGIN_PROLOG

NAME ~DsClCommand

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- GetCategory

```
enum DsShGlobal::DsEShSciCommandCategory GetCategory() const;
```

get the current category BEGIN_PROLOG

NAME GetCategory

DESCRIPTION get the current category for this command

RETURNS the category

END_PROLOG

- GetParametersRef

```
GlParameterList& GetParametersRef();
```

BEGIN_PROLOG

NAME GetParametersRef

DESCRIPTION obtain the current parameters for this command.

RETURNS constant reference to parameters

END_PROLOG

- GetParameters

```
const GlParameterList& GetParameters();
```

the parameters get the current parameters BEGIN_PROLOG

NAME GetParameters

DESCRIPTION obtain the current parameters for this command.

RETURNS constant reference to parameters

END_PROLOG

- GetServiceName

```
const RWCString& GetServiceName() const;
```

the service name get the current service name BEGIN_PROLOG

NAME GetServiceName

DESCRIPTION returns the current service name for this command
RETURNS constant string reference containing service name
END_PROLOG

- operator ==

```
int operator ==(const DsClCommand& ) const;
```

the category comparison operators, always FALSE BEGIN_PROLOG
NAME operator==
DESCRIPTION comparison to another command
RETURNS boolean
END_PROLOG

- operator =

```
DsClCommand& operator =(const DsClCommand& );
```

command to copy assignment operator NOT IMPLEMENTED

- saveOn

```
virtual void saveOn(ostream& os) const;
```

dumps human-readable text of an object BEGIN_PROLOG
NAME saveOn
DESCRIPTION outputs this object in a human-readable fashion
RETURNS nothing
END_PROLOG

- SetCategory

```
void SetCategory(enum DsShGlobal::DsEShSciCommandCategory cat);
```

set the category BEGIN_PROLOG
NAME SetCategory
DESCRIPTION set the category for the command
RETURNS nothing
END_PROLOG

- SetParameters

```
void SetParameters(GlParameterList& pl);
```

set the parameters BEGIN_PROLOG
NAME SetParameters
DESCRIPTION Set the parameters for this command.

RETURNS nothing
END_PROLOG

- SetServiceName

  `void SetServiceName(const RWCString& svc);`

  set the service name BEGIN_PROLOG
  NAME SetServiceName
  DESCRIPTION set the service name for this command
  RETURNS nothing
  END_PROLOG

- Textify

  `virtual RWCString Textify() const;`

  command to compare convert command into text BEGIN_PROLOG
  NAME Textify
  DESCRIPTION converts the command to a text string
  RETURNS the text string
  END_PROLOG

4.3.7.2.2 DsClDescriptor Class

Overview:

Client Public header for the DsClDescriptor class. This class provides an interface for the clients to access ESDT Descriptors on the server

DisTributed Object: Yes

Export Control: Public

Inheritance Relationships:

Attributes:

`myClient: RWCString`

the science data server

`myDataserver: GlUR`

proxy for server-side object

myProxy: DsShDescriptorProxy*

myStatus: GlStatus

the client info

myTypeID: DsShTypeID*

current status

Constructors and Destructor:

DsClDescriptor(const GlUR& dataserver, const RWCString& client, const DsShTypeID& id);

standard constructor

DsClDescriptor(const DsClDescriptor& rhs);

copy constructor - NOT IMPLEMENTED

virtual ~DsClDescriptor();

ESDT type destructor BEGIN_PROLOG

NAME ~DsClDescriptor()

DESCRIPTION The destructor deletes the proxy object via the factory and then deletes the factory. All other memory that was allocated by this object is also deallocated.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- GetMCF

EcUtStatus GetMCF(ostream& os);

BEGIN_PROLOG

NAME GetMCF

DESCRIPTION Get the Metadata configuration file from the sdsrv.

RETURNS Status indicating success or failure

GLOBALS none

FILES none

END_PROLOG

- GetQueryableParameters

```
EcUtStatus GetQueryableParameters(GlParameterList*& pl);
```

BEGIN_PROLOG

NAME GetQueryableParameters

DESCRIPTION Get the queryable parameters for this type from the sdsrv.

RETURNS Status indicating success or failure and a parameterlist with the queryable parameters

GLOBALS none

FILES none

END_PROLOG

- operator =

```
DsClDescriptor& operator =(const DsClDescriptor& rhs);
```

assignment operator - NOT IMPLEMENTED

- saveOn

```
void saveOn(ostream& os) const;
```

dumps human-readable text of an object BEGIN_PROLOG

NAME saveOn

DESCRIPTION Produces human-readable output of an object of this class.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- Validate

```
EcUtStatus Validate(istream& is, GlStatus& stat);
```

BEGIN_PROLOG

NAME Validate

DESCRIPTION Validate the given metadata

RETURNS Status indicating success or failure

GLOBALS none

FILES none

END_PROLOG

4.3.7.2.3 DsClESDTReferenceCollectorVector Class

Overview:

This object is a vector that contains a set of ESDT Reference Collectors. Each ESDT Reference Collector represents a separate connection to a data server. This object is implemented with a Rogue Wave template vector class which means that standard vector services are available on this class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DsClESDTReferenceCollectorVector();`

BEGIN_PROLOG

NAME DsClESDTReferenceCollectorVector

DESCRIPTION

This is the default constructor for this class. It doesn't do much of anything.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`virtual ~DsClESDTReferenceCollectorVector();`

Input. For which client BEGIN_PROLOG

NAME ~DsClESDTReferenceCollectorVector

DESCRIPTION

This is the destructor for this class. It removes all of the ESDT Reference collectors that it has created.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- GetCollector

```
virtual DsClESDTReferenceCollector* GetCollector(const GlUR&
dataserver, const RWCString& client);
```

Default constructor BEGIN_PROLOG

NAME GetCollector

DESCRIPTION

This member function returns an ESDTReferenceCollector that matches the given specification.

INPUTS dataserver - the UR of the dataserver for the desired collector client - the client

OUTPUTS None

RETURNS A pointer to the ESDTReferenceCollector matching the given criteria

GLOBALS None

FILES None

END_PROLOG

4.3.7.2.4 DsClESDTReferenceCollector Class

Overview:

Declarations for the DsClESDTReferenceCollector class.

This object is a vector that contains a set of ESDT Reference Collectors. Each ESDT Reference Collector represents a separate connection to a data server. This object is implemented with a Rogue Wave template vector class which means that standard vector services are available on this class.

Distributed Object: Yes

Export Control: Public

Inheritance Relationships:

Inherits from DsClGenSuspendor

Attributes:

`myERefVectors: RWTPtrOrderedVector<class DsClESDTReferenceVector>`

object from which to assign

Constructors and Destructor:

`DsClESDTReferenceCollector(const DsClESDTReferenceCollector& rhs);`

default constructor

`DsClESDTReferenceCollector();`

the following constructors are defined here (as private) and not implemented in the code, to prevent anyone from using them.

`DsClESDTReferenceCollector(const GlUR& dataserver, const RWCString& clientInfo, unsigned long connectionID);`

constructors

`virtual  ~DsClESDTReferenceCollector();`

specific connection destructor BEGIN_PROLOG

NAME ~DsClESDTReferenceCollector()

DESCRIPTION This function empties the client-side collection of DsClESDTReferences and DsClESDTReferenceVectors prior to deallocating the DsClESDTReferenceCollector object.

RETURNS nothing, destructors can not return anything

END_PROLOG

Operations:

- CreateESDTReference

`GlStatus CreateESDTReference(const GlParameterList& metadata, DsClESDTReferenceVector* ERefVector);`

product type to find end GetERefVector BEGIN_PROLOG

NAME CreateESDTReference(const GlParameterList& metadata, DsClESDTReferenceVector* ERefVector)

DESCRIPTION This function makes a new DsClESDTReference with the provided metadata, and adds it to both the DsClESDTReferenceCollector and the indicated DsClESDTReferenceVector.

RETURNS GlStatus object indicating failure/success and message

END_PROLOG

- GetClient

  `const RWCString& GetClient() const;`

  return server info BEGIN_PROLOG

  NAME GetClient() const

  DESCRIPTION This function provides access to the myClient attribute. This is in support of the DsClESDTReferenceCollectorVector processing.

  RETURNS GlClient reference of the myClient attribute.

  END_PROLOG

- GetDataserver

  `const GlUR& GetDataserver() const;`

  stream on which to write end saveOn BEGIN_PROLOG

  NAME GetDataserver() const

  DESCRIPTION This function provides access to the myDataserver attribute. This is in support of the DsClESDTReferenceCollectorVector processing.

  RETURNS GlUR reference of the myDataserver attribute.

  END_PROLOG

- GetERefVector

  `DsClESDTReferenceVector* GetERefVector(DsClTypeInfo& thisType);`

  commands from server end UpdateState BEGIN_PROLOG

  NAME GetERefVector(DsClTypeInfo& thisType)

  DESCRIPTION This function checks for an existing DsClESDTReferenceVector which matches the provided DsClTypeInfo. If one is found, a pointer to it is returned. If one is not found, a new DsClESDTReferenceVector is created for the new type indicated by DsClTypeInfo and a pointer to the new DsClESDTReferenceCollector is returned. In the case that a new DsClESDTReferenceVector is created, it is assumed that the DsClESDTReferenceCollector (which receives and manages the pointer to the DsClESDTReferenceVector) will delete the heap-storage ERefVector at the appropriate time.

  RETURNS DsClESDTReferenceVector*, pointer to the DsClESDTReferenceVector that matches the information in the provided DsClTypeInfo.

  END_PROLOG

- operator =

  `DsClESDTReferenceCollector& operator =(const DsClESDTReferenceCollector& rhs);`

  object to be copied the following operators are defined here (as private) and not implemented in the code, to prevent anyone from developing them, since they don't actually make sense.

- Reset

  `GlStatus Reset();`

  request to put query in end CreateESDTReference(metadata, ERefVector) BEGIN_PROLOG

  NAME Reset()

  DESCRIPTION This function empties the client-side collection, then creates and submits a request to empty the server-side working collection.

  RETURNS GlStatus object indicating failure/success and message

  END_PROLOG

- saveOn

  `void saveOn(ostream& os) const;`

  destructor operations produce human-readable output BEGIN_PROLOG

  NAME saveOn(ostream& os)

  DESCRIPTION This function produces human-readable formatting of the contents of the object such that it can be used in the overloading of the << operator.

  RETURNS nothing

  END_PROLOG

- Search

  `GlStatus Search(DsClQuery& newQuery);`

  return user info BEGIN_PROLOG

  NAME Search(DsClQuery& newQuery)

  DESCRIPTION This function processes a query. It creates a request from the provided DsClQuery object and submits the request to the server. It processes the results of the request by calling the function which updates the state of the client-side collection to match the working collection on the server side.

  RETURNS GlStatus object indicating failure/success and message

  END_PROLOG

- Search

  `GlStatus Search(DsClQuery& newQuery, DsClRequest* sRp);`

  query to execute

- SetState

  `SetState(Const RWCString);`

- Submit

```
Submit();
```

- UpdateState

```
GlStatus UpdateState(const GlParameterList& requestResults);
```

end Search BEGIN_PROLOG

NAME UpdateState(const GlParameterList& requestResults)

DESCRIPTION This function makes the state of the DsClESDTReferenceCollector match that of the server-side working collection. It does this by processing a list of commands which are returned as the result of some request, most normally a search. In Phase 1 of Release A, only search results are supported.

RETURNS GlStatus object indicating failure/success and message

END_PROLOG

4.3.7.2.5 DsClESDTReferenceVector Class

Overview:

The following classes are defined in this file: DsClESDTReferenceVector ESDT type-level info + ptrs to ESDTs

The following const variables are defined in this file: none

The following defines are defined in this file: #define _DsClESDTReferenceVector_h_ controls includes of this file

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: DsClESDTReferenceVector(void) ; default constructor DsClESDTReferenceVector(construct with data DsClESDTReferenceCollector* collector, DsClTypeInfo& newType) ; ~DsClESDTReferenceVector(void) ; destructor RWBoolean operator==(comparison operator const DsClESDTReferenceVector& rhs) ; saveOn(ostream& os) const; human-readable output GetTypeID() const ; return TypeID DsClTypeInfo& GetTypeInfo(void)const ; return Type-level info SetTypeInfo(put type-level info in const DsClTypeInfo& newType) ; GetCollector() ; return pointer to collector SetCollector(assign collector to this DsClESDTReferenceCollector* collector); DsClESDTReferenceVector(copy constructor const DsClESDTReferenceVector& rhs) ; DsClESDTReferenceVector& operator=(assignment operator const DsClESDTReferenceVector& rhs) ;

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myCollector: DsClESDTReferenceCollector*`

attributes

`myStatus: GlStatus`

`myTypeInfo: DsClTypeInfo`

Constructors and Destructor:

`DsClESDTReferenceVector(const DsClESDTReferenceVector& rhs);`

the copy constructor and operator= are defined here (and not implemented in the code) to prevent anyone from developing them, as they don't make sense constructors

`DsClESDTReferenceVector(DsClESDTReferenceCollector* collector, DsClTypeInfo& newType);`

default constructor

`DsClESDTReferenceVector(void );`

constructors

`virtual  ~DsClESDTReferenceVector(void );`

destructor BEGIN_PROLOG

NAME ~DsClESDTReferenceVector()

DESCRIPTION

Destructor, deallocates the object

RETURNS nothing, destructors can not return anything.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h

END_PROLOG

destructor

Operations:

- GetCollector

  ```
  DsClESDTReferenceCollector* GetCollector();
  ```

  BEGIN_PROLOG

  NAME GetCollector()

  DESCRIPTION

  This function returns the pointer to this object's DsClESDTReferenceCollector.

  RETURNS Pointer to the DsClESDTReferenceCollector associated with this object.

  GLOBALS none

  FILES none

  SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h DsClESDTReferenceCollector.h

  END_PROLOG

- GetTypeID

  ```
  const DsShTypeID& GetTypeID() const;
  ```

  human-readable BEGIN_PROLOG

  NAME GetTypeID() const

  DESCRIPTION

  This function returns the type ID from the attribute myTypeInfo.

  RETURNS The type ID associated with this objects TypeInfo.

  GLOBALS none

  FILES none

  SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h DsClTypeInfo.h

  END_PROLOG

- GetTypeInfo

  ```
  const DsClTypeInfo& GetTypeInfo(void ) const;
  ```

  BEGIN_PROLOG

  NAME GetTypeInfo() const

  DESCRIPTION

  This function returns the type info attribute.

  RETURNS The type-level information associated with this object.

  GLOBALS none

  FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h DsClTypeInfo.h

END_PROLOG

- operator ==

```
int operator ==(const DsClESDTReferenceVector& rhs);
```

operators operators

BEGIN_PROLOG

NAME operator==(const DsClESDTReferenceVector& rhs)

DESCRIPTION

Equality operator, checks whether or not this object has the same values as the rhs object (i.e., deep compare).

RETURNS RWBoolean, true if the two objects match, false otherwise.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h

END_PROLOG

- operator =

```
DsClESDTReferenceVector& operator =(const
DsClESDTReferenceVector& rhs);
```

operators

- saveOn

```
void saveOn(ostream& os) const;
```

operations operations

BEGIN_PROLOG

NAME saveOn(ostream& os) const

DESCRIPTION

This function provides a means to produce human-readable output of the contents of this object. It is expected to work in combination with the overloaded << operator (defined in the .h file)

RETURNS nothing, the output is placed on the ostream provided as a parameter.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h DsClESDTReferenceCollector.h DsClTypeInfo.h DsClESDTReference.h

END_PROLOG

- SetCollector

```
GlStatus SetCollector(DsClESDTReferenceCollector* collector);
```

BEGIN_PROLOG

NAME SetCollector()

DESCRIPTION

This function sets this object's myCollector attribute to point to the provided DsClESDTReferenceCollector.

RETURNS GlStatus object indicating whether or not the operation completed successfully.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h DsClESDTReferenceCollector.h GlStatus.h

END_PROLOG

- SetTypeInfo

```
GlStatus SetTypeInfo(const DsClTypeInfo& newType);
```

BEGIN_PROLOG

NAME SetTypeInfo(const DsClTypeInfo& newType)

DESCRIPTION

This function sets the type info attribute to the provided data.

RETURNS GlStatus, indicating whether or not the operation was successfully completed.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClESDTReferenceVector.h DsClTypeInfo.h GlStatus.h

END_PROLOG

4.3.7.2.6 DsClESDTReference Class

Overview:

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

Declarations for the DsClESDTReferenceCollector class.

This object is a vector that contains a set of ESDT Reference Collectors. Each ESDT Reference Collector represents a separate connection to a data server. This object is implemented with a

Rogue Wave template vector class which means that standard vector services are available on this class.

The following classes are defined in this file: DsClESDTReferenceVector ESDT type-level info + ptrs to ESDTs

The following const variables are defined in this file: none

The following defines are defined in this file: #define _DsClESDTReferenceVector_h_ controls includes of this file

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: DsClESDTReferenceVector(void) ; default constructor DsClESDTReferenceVector(construct with data DsClESDTReferenceCollector* collector, DsClTypeInfo& newType) ; ~DsClESDTReferenceVector(void) ; destructor RWBoolean operator==(comparison operator const DsClESDTReferenceVector& rhs) ; saveOn(ostream& os) const; human-readable output GetTypeID() const ; return TypeID DsClTypeInfo& GetTypeInfo(void)const ; return Type-level info SetTypeInfo(put type-level info in const DsClTypeInfo& newType) ; GetCollector() ; return pointer to collector SetCollector(assign collector to this DsClESDTReferenceCollector* collector); DsClESDTReferenceVector(copy constructor const DsClESDTReferenceVector& rhs) ; DsClESDTReferenceVector& operator=(assignment operator const DsClESDTReferenceVector& rhs) ;

Distributed Object: No

Export Control: Public

Inheritance Relationships:

Attributes:

`myCachedAttributes: GlParameterList`

UR for this Reference

`myCollector: DsClESDTReferenceCollector*`

state of completion

`myCompleteFlag: int`

object with type level info

`myReferenceVector: DsClESDTReferenceVector*`

Collector for this Reference

`myStatus: GlStatus`

Cached metadata attributes

`myTypeInfo: const DsClTypeInfo*`

Status of this object

`myUR: GlUR`

attributes

`ourCollectorVector: DsClESDTReferenceCollectorVector`

Input. Type level info vector

Constructors and Destructor:

`DsClESDTReference(const GlParameterList& pars, DsClESDTReferenceCollector* ec, DsClESDTReferenceVector* erv);`

constructor used by the ESDTReferenceCollector

`DsClESDTReference(const DsClESDTReference& ref);`

User profile information BEGIN_PROLOG

NAME DsClESDTReference

DESCRIPTION This is the copy constructor for the DsClESDTReference.

INPUTS ref - the ESDT Reference to make a copy of

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

NOTES

END_PROLOG

`DsClESDTReference();`

Vector Reference belongs to default constructor defined here to prevent anyone from using it

`DsClESDTReference(const GlUR& ur, const GlUR& dss, const RWCString& me);`

constructors not implemented yet

`virtual  ~DsClESDTReference();`

Get my collector BEGIN_PROLOG

NAME ~DsClESDTReference

DESCRIPTION This is the destructor for the ESDT Reference. It destroys the list of cached attributes for this ESDT Reference.

INPUTS None

OUTPUTS None

RETURNS None

GLOBALS None

FILES None

NOTES

END_PROLOG

Operations:

- ClearCompleteFlag

```
void ClearCompleteFlag();
```

Flag as complete BEGIN_PROLOG

NAME ClearCompleteFlag

DESCRIPTION This member function is used to clear the flag that indicates that this ESDT Reference is complete. If this function is used, the ESDT reference is considered to be incomplete.

INPUTS None

OUTPUTS None

RETURNS None

GLOBALS None

FILES

NOTES

END_PROLOG

- DoInspectCmd

```
virtual const GlStatus& DoInspectCmd(GlParameterList&
nonCachedParms, GlParameterList& resultsSet);
```

Output. ESDT Results BEGIN_PROLOG

NAME DoInspectCmd

DESCRIPTION This member function creates an inspect command with the given parameters and submits it.

INPUTS nonCachedParms - a parameterList with a parameter for each of the metadata attributes to get the values for resultsSet - a reference to the parameterList that will have the results of the inspect command OUTPUTS resultsSet parameter list containing a parameter for each metadata attribute

RETURNS Status indicating whether the inspect was successful or not.

GLOBALS None

FILES None

NOTES

DETAILS

END_PROLOG

- Empty

```
virtual void Empty();
```

Input. Ref Vector for this BEGIN_PROLOG

NAME Empty

DESCRIPTION This member function clears out all of the attributes of this ESDT Reference except for the collector, eref vector, and typeinfo.

INPUTS None

OUTPUTS None

RETURNS None

GLOBALS None

FILES None

NOTES

END_PROLOG

- Fill

```
virtual const GlStatus& Fill(const GlParameterList& pars,
DsClESDTReferenceVector* erv);
```

Input. Initialize from BEGIN_PROLOG

NAME Fill

DESCRIPTION This member function fills in the attributes of this ESDT Reference with the provided values. Once the attributes have been set, the ESDT Reference is complete. INPUTS pars - A parameter list which contains the set of attributes to be cached for this ESDT. erv - A pointer to the ESDT Reference Vector for this ESDT. The ESDT Reference Vector maintains an association between this ESDT Reference and its set of type-level information.

OUTPUTS None

RETURNS Status indicating whether the ESDT Reference was successfully filled. Status can be FAILURE if the given parameter list is missing the "ur" parameter.

GLOBALS None

FILES None

NOTES

END_PROLOG

- GetCollector

```
DsClESDTReferenceCollector& GetCollector();
```

Get status of this instance BEGIN_PROLOG

NAME GetCollector

DESCRIPTION This member function returns the non-const ESDT Collector of this ESDT Reference.

INPUTS None

OUTPUTS None

RETURNS DsClESDTReferenceCollector that contains this ESDT Reference

GLOBALS None

FILES None

END_PROLOG

- GetCollector

```
DsClESDTReferenceCollector& GetCollector() const;
```

Input/Output. Send msgs to BEGIN_PROLOG

NAME GetCollector

DESCRIPTION This member function returns the ESDT Collector of this ESDT Reference.

INPUTS None

OUTPUTS None

RETURNS const DsClESDTReferenceCollector that contains this ESDT Reference

GLOBALS None

FILES None

END_PROLOG

- GetESDTResults

```
virtual const GlStatus& GetESDTResults(DsClRequest& req,
GlParameterList& ESDTresults);
```

Clear objects attributes BEGIN_PROLOG

NAME GetESDTResults

DESCRIPTION This member function finds the results of the given ESDT request

INPUTS req - a request that was previously submitted ESDTResults - a reference to the results

OUTPUTS ESDTResults parameter list containing the results of an ESDT request

RETURNS Status indicating whether the results were located or not.

GLOBALS None

FILES None

NOTES

DETAILS

END_PROLOG

- GetReferenceVector

```
DsClESDTReferenceVector* const GetReferenceVector() const;
```

hidden get functions BEGIN_PROLOG

NAME GetReferenceVector

DESCRIPTION This member function returns a pointer to the vector that contains all ESDT References of this type

INPUTS None

OUTPUTS None

RETURNS const DsClESDTReference Vector pointer for object that contains type level information applicable to this ESDT Reference.

GLOBALS None

FILES

NOTES

An example of type level information might be the set of metadata. All ESDT references of the same type have the same set of metadata attributes (only the values can be different)

END_PROLOG

- GetStatus

```
virtual const GlStatus& GetStatus() const;
```

Get Type Info object BEGIN_PROLOG

NAME GetStatus

DESCRIPTION This member function returns the status of this ESDT Reference.

INPUTS None

OUTPUTS None

RETURNS const GlStatus indicating status of this instance.

GLOBALS None

FILES None

END_PROLOG

- GetTypeID

```
virtual const DsShTypeID& GetTypeID() const;
```

Get UR BEGIN_PROLOG

NAME GetTypeID

DESCRIPTION This member function returns the object that contains the specification for this ESDTs type.

INPUTS None

OUTPUTS None

RETURNS A DsShTypeID reference which contains this references type information.

GLOBALS None

FILES None

NOTES

END_PROLOG

- GetTypeInfo

```
const DsClTypeInfo& GetTypeInfo() const;
```

Get Reference Vector object BEGIN_PROLOG

NAME GetTypeInfo

DESCRIPTION This member function returns a pointer to the object that contains type level information for this ESDT Reference

INPUTS None

OUTPUTS None

RETURNS const DsClTypeInfo pointer for object that contains type level information applicable to this ESDT Reference.

GLOBALS None

FILES

NOTES

An example of type level information might be the set of metadata. All ESDT references of the same type have the same set of metadata attributes (only the values can be different)

END_PROLOG

- GetUR

```
const GlUR& GetUR() const;
```

Input. right-hand side value BEGIN_PROLOG

NAME GetUR

DESCRIPTION This member function returns the UR associated with this ESDT Reference.

INPUTS None

OUTPUTS None

RETURNS const GlUR

GLOBALS None

FILES None

END_PROLOG

- Init

```
void Init(const DsClESDTReference& ref);
```

Flag as incomplete hidden member functions BEGIN_PROLOG

NAME Init

DESCRIPTION This member function combines the common aspects of copying and assigning ESDT References to each other. This was to minimize duplication of code.

INPUTS ref - the ESDT Reference that is the source information from which "this" ESDT Reference is populated.

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

NOTES

END_PROLOG

- Inspect

```
virtual const GlStatus& Inspect(const RWTPtrOrderedVector<class
RWCString> attributes, GlParameterList& resultsParms);
```

Input-output. Parameter with results BEGIN_PROLOG

NAME Inspect

DESCRIPTION This member function provides metadata attribute values for this ESDT Reference based on the given vector of attribute names.

INPUTS attributes - a vector containing strings with attribute names for the metadata attributes to obtain resultsParms - a parameterList in which to fill in the obtained attributes and values. OUTPUTS resultsParms containing the desired attributes and associated values

RETURNS Status indicating whether the inspect was successful or not.

GLOBALS None

FILES None

NOTES In the future, this service will validate the requested attributes against the queryable parameters. Also, when getQP exists, this will probably be rewritten to create a parameterList from the given attribute vector and calling the other version of this service

DETAILS Check to see if the requested attributes are valid by looking at queryable parameters. If any attributes are invalid, return immediately with an indication of which attributes are invalid. Next, see if the desired attributes exist on the client side in the cached attributes. For any

that do not exist, formulate a request that asks the server for this information. Add any newly obtained attributes to the cached list for this ESDT Reference.

END_PROLOG

- Inspect

```
virtual const GlStatus& Inspect(GlParameterList& resultsParms);
```

Get Type ID BEGIN_PROLOG

NAME Inspect

DESCRIPTION This member function provides metadata attribute values for this ESDT Reference based on the given parameter list.

INPUTS resultsParms - a parameterList in which to fill in the obtained attributes and values. OUTPUTS resultsParms containing the desired attributes and associated values

RETURNS Status indicating whether the inspect was successful or not.

GLOBALS None

FILES None

NOTES The caller of this routine will need to recursively delete the results of this operation. In the future, this service will validate the requested attributes against the queryable parameters.

DETAILS Check to see if the requested attributes are valid by looking at queryable parameters. If any attributes are invalid, return immediately with an indication of which attributes are invalid. Next, see if the desired attributes exist on the client side in the cached attributes. For any that do not exist, formulate a request that asks the server for this information. Add any newly obtained attributes to the cached list for this ESDT Reference.

END_PROLOG

- IsComplete

```
virtual const int& IsComplete() const;
```

Input. Request to be submitted BEGIN_PROLOG

NAME IsComplete

DESCRIPTION This member function provides an indication of whether the ERef has completed construction or not. No services can be performed on an incomplete ESDT Reference.

INPUTS None

OUTPUTS None

RETURNS RWBoolean indicating whether the ESDT Reference has or has not been completed.

GLOBALS None

FILES

NOTES

END_PROLOG

- operator ==

  `int operator ==(const DsClESDTReference& rhs) const;`

  Input. old value BEGIN_PROLOG

  NAME operator==

  DESCRIPTION This member function tests equality of two ESDT References

  INPUTS Two ESDT References

  OUTPUTS None

  RETURNS RWBoolean indicating whether the given ESDT References are equal.

  GLOBALS None

  FILES None

  END_PROLOG

- operator =

  `DsClESDTReference& operator =(const DsClESDTReference& ref);`

  copy constructor operators BEGIN_PROLOG

  NAME operator=

  DESCRIPTION This is the assignment operator for the DsClESDTReference.

  INPUTS ref - the ESDT Reference that is on the right hand side of an assignment statement.

  OUTPUTS None

  RETURNS N/A

  GLOBALS None

  FILES None

  NOTES

  END_PROLOG

- saveOn

  `void saveOn(ostream& os) const;`

  Make readable string BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION This member function is used to output this ESDT Reference to a stream. It is used primarily for debug purposes so that you can see the contents of an ESDT Reference during a debug session.

  INPUTS os - an output stream on which to write this ESDT Reference

  OUTPUTS None

  RETURNS None

  GLOBALS None

FILES

NOTES

END_PROLOG

- SetCompleteFlag

```
void SetCompleteFlag();
```

Input. Ptr to erefVector BEGIN_PROLOG

NAME SetCompleteFlag

DESCRIPTION This member function is used to set the flag that indicates that this ESDT Reference has completely constructed.

INPUTS None

OUTPUTS None

RETURNS None

GLOBALS None

FILES

NOTES

END_PROLOG

- SetReferenceVector

```
void SetReferenceVector(DsClESDTReferenceVector* rv);
```

Input. Has type-level info BEGIN_PROLOG

NAME SetReferenceVector

DESCRIPTION This member function assigns the pointer that contains all ESDT References of this type.

INPUTS DsClESDTReferenceVector pointer

OUTPUTS None

RETURNS None

GLOBALS None

FILES

NOTES

END_PROLOG

- SetTypeInfo

```
void SetTypeInfo(DsClTypeInfo* ti);
```

Input. UR BEGIN_PROLOG

NAME SetTypeInfo

DESCRIPTION This member function is used to assign this References type level object pointer. This is a protected function.

INPUTS ti - A pointer to a DsClTypeInfo object

OUTPUTS None

RETURNS None

GLOBALS None

FILES

NOTES

END_PROLOG

- SetUR

```
void SetUR(const GlUR& ur);
```

Get Collector object hidden set member functions BEGIN_PROLOG

NAME SetUR

DESCRIPTION This member function is used to set the URs value. It is protected.

INPUTS ur - A reference to a UR object

OUTPUTS None

RETURNS None

GLOBALS None

FILES

NOTES

END_PROLOG

- Submit

```
virtual const GlStatus& Submit(DsClRequest& req);
```

Output. Inspect results BEGIN_PROLOG

NAME Submit

DESCRIPTION This member function ensures that the given request is submitted to the Data Server that holds this ESDT Reference.

INPUTS req - a request to be processed by this ESDT References data server

OUTPUTS None

RETURNS Status indicating whether the request was successful or not

GLOBALS None

FILES None

NOTES

END_PROLOG

- Textify

  `virtual RWCString Textify() const;`

  Is Reference Complete BEGIN_PROLOG

  NAME Textify

  DESCRIPTION This member function is used to obtain a textstring whose contents are an ASCII representation of the attributes cached for this ESDT Reference.

  INPUTS None

  OUTPUTS None

  RETURNS

  An RWCString which contains a list of all of the cached attributes in a readable/displayable form (i.e. a textstring)

  GLOBALS None

  FILES

  NOTES

  END_PROLOG

4.3.7.2.7 DsClGenConnector Class

Overview:

Client Public header for the DsClGenConnector class, which is a base class for other client-side connectors. This class encapsulates communications with a data server

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myClient: RWCString`

the server

`myConnection: DsShConnectionProxy*`

`myDataserver: GlUR`

associated connection object

myStatus: GlStatus

the client info

Constructors and Destructor:

DsClGenConnector(const GlUR& dataserver, const RWCString& client, unsigned long connID);

standard constructor

DsClGenConnector(const DsClGenConnector& rhs);

so it can SubmitToServer() copy constructor - NOT IMPLEMENTED

virtual ~DsClGenConnector();

re-connect to this ID destructor BEGIN_PROLOG

NAME ~DsClGenConnector()

DESCRIPTION This function deallocates the DsClGenConnector object.

RETURNS nothing

END_PROLOG

Operations:

- Connect

virtual void Connect();

actually connect to server BEGIN_PROLOG

NAME Connect()

DESCRIPTION This function establishes the connection to the server. This involves creating a distributed object which communicates with the real server on a different machine

RETURNS nothing

END_PROLOG

- Disconnect

virtual void Disconnect();

disconnect from the server BEGIN_PROLOG

NAME Disconnect()

DESCRIPTION This function destroys the connection to the server.

RETURNS nothing

END_PROLOG

- GetConnectionID

```
virtual unsigned long GetConnectionID();
```

obtain connection ID BEGIN_PROLOG

NAME GetConnectionID()

DESCRIPTION This function queries the server for the ID of the session which has been established.

RETURNS DsEShConnectionID

END_PROLOG

- GetServer

```
const GlUR& GetServer() const;
```

obtain the server

- operator =

```
DsClGenConnector& operator =(const DsClGenConnector& rhs);
```

assignment operator - NOT IMPLEMENTED

- saveOn

```
virtual void saveOn(ostream& os) const;
```

dumps human-readable text of an object BEGIN_PROLOG

NAME saveOn

DESCRIPTION Produces human-readable output of an object of this class.

RETURNS nothing

END_PROLOG

- SetClient

```
void SetClient(RWCString& Client);
```

assign client info BEGIN_PROLOG

NAME SetClient

DESCRIPTION assigns the client info object for this collector

RETURNS nothing

END_PROLOG

- SetDataserver

```
void SetDataserver(GlUR& dataserver);
```

assign which data server BEGIN_PROLOG

NAME SetDataServer

DESCRIPTION assigns the data server for this collector

RETURNS nothing

END_PROLOG

- SubmitToServer

  `virtual GlStatus SubmitToServer(DsShSRequestProxy& srp);`

  internal submit function BEGIN_PROLOG

  NAME SubmitToServer

  DESCRIPTION Protected function that is used internally to submit a request to the server and obtain the results (of the submission, but not necessarily of the execution of the request).

  RETURNS status of the submission

  END_PROLOG

4.3.7.2.8 DsClGenRequestInt Class

Overview:

The definition of the DsClGenRequestInt class, which represents the generic portion of the client-sides interface to request objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShUpdateable

Attributes:

`myCallbackData: void*`

status callback pointer

`myCallback: void (*) (class DsClGenRequestInt*, void*)`

true -> request has been submitted

`myResults: GlParameterList*`

submitted component (after submission)

`mySRequest: DsShSRequestProxy*`

`myStatus: GlStatus`

holder for results

mySubmittedFlag: int

most recent status

Constructors and Destructor:

DsClGenRequestInt(const DsClGenRequestInt&);

the first (and maybe only) command copy constructor

DsClGenRequestInt(DsShGenCommandImp* cmd);

normal constructor

DsClGenRequestInt();

default constructor

virtual ~DsClGenRequestInt();

request to assign destructor BEGIN_PROLOG

NAME ~DsClGenRequestInt

DESCRIPTION destructor

END_PROLOG

Operations:

- AsyncUpdate

void AsyncUpdate();

called on every asynchronous status update BEGIN_PROLOG

NAME AsyncUpdate

DESCRIPTION called for each asynchronous status update

RETURNS nothing

END_PROLOG

- Cancel

virtual GlStatus Cancel();

request to compare cancel an executing request BEGIN_PROLOG

NAME Cancel

DESCRIPTION Used to cancel an active request.

RETURNS Status indicating success or failure.

NOTES NOT IMPLEMENTED yet (requests are still synchronous).

END_PROLOG

- GenSubmit

```
virtual GlStatus GenSubmit(DsClGenConnector& connector);
```

submits to the connector BEGIN_PROLOG

NAME GenSubmit

DESCRIPTION Used to submit a request to the given connector/dataserver.

RETURNS status

END_PROLOG

- GetCommandImp

```
virtual const DsShGenCommandImp* GetCommandImp(unsigned int idx)
const = 0;
```

get a pointer to a command imp from the derived object

- GetRequestImp

```
virtual DsShGenRequestImp* GetRequestImp() = 0;
```

the collector/data server get a pointer to the request implementation object

- GetRequestImp

```
virtual const DsShGenRequestImp* GetRequestImp() const = 0;
```

get a pointer to the request implementation object (const version)

- GetResults

```
virtual const GlParameterList& GetResults() const;
```

obtain current results value for this request BEGIN_PROLOG

NAME GetResults

DESCRIPTION Used to obtain the current results of execution of the request.

RETURNS reference to constant parameter list of results

NOTES Since the results are contained in a GlParameterList, there is a large range of possible values. GlParameterList's can contain many values of various types.

END_PROLOG

- GetStatus

```
const GlStatus& GetStatus() const;
```

obtain current status of this request BEGIN_PROLOG

NAME GetStatus

DESCRIPTION Used to obtain the current status of this request. It is updated asynchronously as the request executes on the server.

RETURNS constant reference to the status object

END_PROLOG

- operator ==

  ```
  int operator ==(const DsClGenRequestInt& ) const;
  ```

  comparison operator BEGIN_PROLOG

  NAME operator==

  DESCRIPTION comparison operator

  RETURNS boolean

  END_PROLOG

- operator =

  ```
  DsClGenRequestInt& operator =(const DsClGenRequestInt& );
  ```

  request to copy assignment operator

- saveOn

  ```
  virtual void saveOn(ostream& os) const;
  ```

  dumps human-readable text of an object BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION dumps human-readable text of an object

  RETURNS nothing

  END_PROLOG

- SetStatusCallback

  ```
  void SetStatusCallback(void (*) (class DsClGenRequestInt*,
  void*) cb, void* vp);
  ```

  setup a callback, which makes this an asynchronous requesty BEGIN_PROLOG

  NAME SetStatusCallback

  DESCRIPTION Setup a callback, which makes this an asynchronous request

  RETURNS nothing

  END_PROLOG

4.3.7.2.9 DsClGenSuspendor Class

Overview:

Client Public header for the DsClGenSuspendor class, which is a base class for specialized collector types requiring state management. This class encapsulates communications with a data server

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsClGenConnector

Attributes:

Constructors and Destructor:

```
DsClGenSuspendor(const GlUR& dataserver, const RWCString&
client, unsigned long connID);
```

standard constructor

```
DsClGenSuspendor(const DsClGenSuspendor& rhs);
```

copy constructor - NOT IMPLEMENTED

```
virtual  ~DsClGenSuspendor();
```

re-connect to this ID destructor BEGIN_PROLOG

NAME ~DsClGenSuspendor()

DESCRIPTION This function deallocates the DsClGenSuspendor object.

RETURNS nothing

END_PROLOG

Operations:

- operator =

```
DsClGenSuspendor& operator =(const DsClGenSuspendor& rhs);
```

assignment operator - NOT IMPLEMENTED

- Resume

```
virtual GlStatus Resume();
```

BEGIN_PROLOG

NAME Suspend

DESCRIPTION In phase 3, this will resume a suspended connection

RETURNS status

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

dumps human-readable text of an object BEGIN_PROLOG

NAME saveOn

DESCRIPTION Produces human-readable output of an object of this class.

RETURNS nothing

END_PROLOG

- Suspend

```
virtual GlStatus Suspend();
```

any output stream suspend and resume features - NOT IMPLEMENTED UNTIL PHASE 3 BEGIN_PROLOG

NAME Suspend

DESCRIPTION In phase 3, this will suspend an active connection

RETURNS status

END_PROLOG

4.3.7.2.10 DsClInsertCommand Class

Overview:

DsClInsertCommand provides an encapsulated mechanism for transporting ESDT insert command parameters through the SDSRV client interface.

The following classes are defined in this file: DsClInsertCommand

The following const variables are defined in this file: none

The following defines are defined in this file: DsClInsertCommandH ensures file isn't read twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No Provides basic CSDT functionality

Export Control: Implementation

Inheritance Relationships:

Inherits from DsClCommand

Attributes:

```
myServiceParams: GlParameterList&
```

```
mySubType: RWCString
```

```
myVersion: RWCString
```

```
scienceGrpCreated: int
```

Constructors and Destructor:

```
DsClInsertCommand(DsShTypeID& typeID, enum
DsClInsertCommand::DsClInsertCommand_h_00010 groupToCreate,
const RWCString& metadatafile, const RWTPtrOrderedVector<class
RWCString>& datafiles);
```

```
DsClInsertCommand(DsShTypeID& typeID);
```

```
DsClInsertCommand();
```

Constructors

```
virtual  ~DsClInsertCommand();
```

destructor BEGIN_PROLOG

NAME ~DsClInsertCommand

DESCRIPTION Destructor for DsClInsertCommand

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- AddDataFiles

```
EcUtStatus AddDataFiles(GlParameterList* groupToAddTo, const
RWTPtrOrderedVector<class RWCString>& datafiles, const
RWTPtrOrderedVector<class RWCString>& types);
```

Private methods

BEGIN_PROLOG

NAME AddDataFiles

DESCRIPTION Add data files & (if present) file type to group

INPUTS const RWTPtrOrderedVector<RWCString>& - data files to add GlParameterList& groupToAddTo - the group to add file too

OUTPUTS none

RETURNS EcUtStatus - SUCCESS or FAILURE

END_PROLOG

- AddGroup

```
EcUtStatus AddGroup(enum
DsClInsertCommand::DsClInsertCommand_h_00010 groupToCreate,
const RWCString& metadatafile, const RWTPtrOrderedVector<class
RWCString>& datafiles, const RWTPtrOrderedVector<class
RWCString>& types);
```

Add a group with actual files and associated file content types. The number of types must match the number of datafiles BEGIN_PROLOG

NAME AddGroup

DESCRIPTION Create a granule group, with metadata file & file types

INPUTS DsEClGroupType groupToCreate, const RWCString& metadatafile, const RWTPtrOrderedVector<RWCString>& datafiles, const RWTPtrOrderedVector<RWCString>& types

OUTPUTS none

RETURNS EcUtStatus - OK or FAILED

END_PROLOG

- AddGroup

```
EcUtStatus AddGroup(const RWTPtrOrderedVector<class RWCString>&
myURs, enum DsClInsertCommand::DsClInsertCommand_h_00010
groupToCreate);
```

Add a set of URs BEGIN_PROLOG

NAME AddGroup

DESCRIPTION Create a granule group, with metadata file & file types

INPUTS DsEClGroupType groupToCreate, const RWCString& metadatafile, const RWTPtrOrderedVector<RWCString>& datafiles, const RWTPtrOrderedVector<RWCString>& types

OUTPUTS none

RETURNS EcUtStatus - OK or FAILED

END_PROLOG

- AddGroup

```
EcUtStatus AddGroup(enum
DsClInsertCommand::DsClInsertCommand_h_00010 groupToCreate,
const RWTPtrOrderedVector<class RWCString>& datafiles);
```

Add a group with data files, no metadata file Does not accept Science group - all science groups must have one metadata file. BEGIN_PROLOG

NAME AddGroup

DESCRIPTION Create a granule group, no metadata file

INPUTS DsEClGroupType - the group type to add const RWTPtrOrderedVector<RWCString>& datafiles - list of datafiles

OUTPUTS none

RETURNS EcUtStatus - OK or FAILED

END_PROLOG

- AddGroup

```
EcUtStatus AddGroup(enum
DsClInsertCommand::DsClInsertCommand_h_00010 groupToCreate,
const RWCString& metadatafile, const RWTPtrOrderedVector<class
RWCString>& datafiles);
```

N.B. Only ONE science group may be created. These interfaces do NOT guarantee that an insert signature will be correct for any one particular datatype. For example: the 'aType' ESDT Insert must have a metadata file provided with a browse file - use of the second interface type to create a browse group would result in Insert failure. How do you know what is expected as input for a particular datatype ? - use the Advertising service. Add a group with one metadata file & one or more data files BEGIN_PROLOG

NAME AddGroup

DESCRIPTION Create a granule group with metadata file

INPUTS DsEClGroupType - the group type to add const RWCString& metadatafile - name of metadata file const RWTPtrOrderedVector<RWCString>& datafiles - list of datafiles

OUTPUTS none

RETURNS EcUtStatus - SUCCESS or FAILURE

END_PROLOG

- AddMetadata

```
EcUtStatus AddMetadata(GlParameterList* groupToAddTo, const
RWCString& metadatafile);
```

BEGIN_PROLOG

NAME AddMetadata

DESCRIPTION Add metadata file to group

INPUTS const RWCString& metadatafile - the file to add GlParameterList& groupToAddTo - the group to add file too

OUTPUTS none

RETURNS EcUtStatus - SUCCESS or FAILURE

END_PROLOG

- GetAGroup

```
EcUtStatus GetAGroup(enum
DsClInsertCommand::DsClInsertCommand_h_00010 groupToCreate,
GlParameterList* newGroup);
```

BEGIN_PROLOG

NAME GetAGroup

DESCRIPTION Create a parameterlist with correct name, subtype & version

INPUTS DsEClGroupType - the group type to add GlParameterList* newGroup - returned parameterList

OUTPUTS none

RETURNS EcUtStatus - OK or FAILURE

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

- SetDataType

```
void SetDataType(DsShTypeID& typeID);
```

BEGIN_PROLOG

NAME SetDataType

DESCRIPTION Set the data type of data to insert

INPUTS DsShTypeID& typeID

OUTPUTS none

RETURNS void

END_PROLOG

4.3.7.2.11 DsClQuery Class

Overview:

The following classes are defined in this file: DsClQuery convenient way for clients to make search requests

The following const variables are defined in this file: none

The following defines are defined in this file: _DsClQuery_h_ ensures file isn't read twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: DsClQuery() default constructor DsClQuery(const GlParameterList& constraints) full constructor DsClQuery(const DsClQuery& rhs) copy constructor ~DsClQuery() destructor operator=(const DsClQuery& rhs) assignment operator operator==(const DsClQuery& rhs) const equality operator saveOn(ostream& os) const; readable output SetQueryType(DsSrCommandBase::DsESrCommandCategory type) sets data member DsSrCommandBase::DsESrCommandCategory GetQueryType() gets data member SetConstraints(const GlParameterList& newConstraints) sets data member GetConstraints(GlParameterList& constraints) gets data member ConvertToCommand(DsClCommand& queryCommand) creates DsClCommand object from its data members

Distributed Object: No

Export Control: Public

Inheritance Relationships:

Attributes:

`myConstraints: GlParameterList`

currently CATALOG presumably someday VALIDS as well

`myQueryType: enum DsShGlobal::DsEShSciCommandCategory`

attributes

`myStatus: GlStatus`

Constructors and Destructor:

```
DsClQuery(const DsClQuery& rhs);

DsClQuery(const GlParameterList& constraints);

DsClQuery();
```

constructors

```
virtual  ~DsClQuery();
```

destructor destructor

BEGIN_PROLOG

NAME ~DsClQuery()

DESCRIPTION

This function deallocates the DsClQuery object.

RETURNS nothing, destructors can not return anything

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h

END_PROLOG

Operations:

- ConvertToCommand

```
GlStatus ConvertToCommand(DsClCommand& queryCommand);
```

to use ConvertToCommand BEGIN_PROLOG

NAME ConvertToCommand DsClCommand& queryCommand)

DESCRIPTION

This function sets up the provided DsClCommand (via a reference to it) with the necessary name/category/parameters to be a search command.

RETURNS GlStatus object indicating succes or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h GlParameterList.h GlStatus.h DsClCommand.h

END_PROLOG

- GetConstraints

```
GlStatus& GetConstraints(GlParameterList& constraints);
```

BEGIN_PROLOG

NAME GetConstraints()

DESCRIPTION

This function puts the reference to the GlParameterList which is stored in attribute myConstraints into the provided GlParameterList reference variable (the output parameter on

the function call). The actual parameter list is managed elsewhere, this operation just hands over the reference to the actual parameter list.

RETURNS GlStatus object indicating succes or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h GlParameterList.h GlStatus.h

END_PROLOG

- GetQueryType

```
enum DsShGlobal::DsEShSciCommandCategory GetQueryType(void );
```

BEGIN_PROLOG

NAME GetQueryType()

DESCRIPTION

This function returns the value of the myQueryType attribute.

RETURNS DsEShCommandCategory, the type of this query.

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h DsShCommandBase.h

END_PROLOG

- operator ==

```
int operator ==(const DsClQuery& rhs) const;
```

BEGIN_PROLOG

NAME operator==(const DsClQuery& rhs) const

DESCRIPTION

This function checks to see if this object is content-equivalent to the provided DsClQuery object.

RETURNS RWBoolean, TRUE if the objects are content-equivalent, FALSE otherwise

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h GlParameterList.h GlStatus.h

END_PROLOG

- operator =

```
DsClQuery& operator =(const DsClQuery& rhs);
```

operators operators

BEGIN_PROLOG

NAME operator=(const DsClQuery& rhs)

DESCRIPTION

This function makes this object an exact copy of the provided object.

RETURNS DsClQuery&, reference to self

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h GlParameterList.h GlStatus.h

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

operations operations

BEGIN_PROLOG

NAME saveOn(ostream& os)

DESCRIPTION

This function produces human-readable output of this object's data. It is intended to work in conjunction with the overloaded operator<< defined in the DsClQuery.h file.

RETURNS nothing, the output is put on the provided ostream.

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h GlParameterList.h GlStatus.h

END_PROLOG

- SetConstraints

```
GlStatus& SetConstraints(const GlParameterList& newConstraints);
```

BEGIN_PROLOG

NAME SetConstraints(const GlParameterList& newConstraints)

DESCRIPTION

This function assigns the provided parameter list reference to the myConstraints attribute. Note that the parameter list of constraints is managed elsewhere. This object manages a reference to the actual parameter list.

RETURNS GlStatus object indicating success or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h GlParameterList.h GlStatus.h

END_PROLOG

- SetQueryType

`GlStatus& SetQueryType(enum DsShGlobal::DsEShSciCommandCategory newType);`

human-readable BEGIN_PROLOG

NAME SetQueryType(DsShCommandBase::DsEShCommandCategory newType)

DESCRIPTION

This function assigns the provided value to the myQueryType attribute.

RETURNS GlStatus object, indicating success or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS The following include files are required for this function: DsClQuery.h GlStatus.h DsShCommandBase.h

END_PROLOG

4.3.7.2.12 DsClRequest Class

Overview:

Declaration of the DsClRequest class. This is the object that clients submit to a data server to do anything. It is composed of a sequence of DsClCommand objects (created by the client), which are the basic execution units.

Declaration of the DsClRequestExample class. This is the object that

NOTES

Distributed Object: No

Export Control: Public

Inheritance Relationships:

Inherits from DsClGenRequestInt

Attributes:

`mySciImp: DsShSciRequestImp*`

request to assign

Constructors and Destructor:

```
DsClRequest(const DsClRequest& );
```

copy constructor NOT IMPLEMENTED

```
DsClRequest(DsClCommand& cmd, enum DsShGlobal::DsEShSciPriority
pty);
```

normal constructor

```
DsClRequest();
```

default constructor

```
virtual  ~DsClRequest();
```

priority of this request destructor BEGIN_PROLOG

NAME ~DsClRequest

DESCRIPTION destructor

END_PROLOG

Operations:

- Disconnect

```
Disconnect();
```

- GetCommandImp

```
virtual const DsShGenCommandImp* GetCommandImp(unsigned int idx)
const;
```

get a pointer to a command imp from the derived object BEGIN_PROLOG

NAME GetCommandImp

DESCRIPTION Used to obtain the implementation of the command at the give idx.

RETURNS pointer to the command implementation or rwnil if idx is out of range

END_PROLOG

- GetDomain

```
const RWTPtrOrderedVector<class GlUR>& GetDomain() const;
```

obtain the current domain for the request BEGIN_PROLOG

NAME GetDomain

DESCRIPTION obtains the current domain of the request

RETURNS the domain

END_PROLOG

- GetPriority

  `enum DsShGlobal::DsEShSciPriority GetPriority() const;`

  obtain the current priority of the request BEGIN_PROLOG

  NAME GetPriority

  DESCRIPTION obtains the priority of the request

  RETURNS the priority

  END_PROLOG

- GetQuery

  `DsClQuery* GetQuery();`

  the query obtain the query associated with this request

- GetRequestImp

  `virtual const DsShGenRequestImp* GetRequestImp() const;`

  obtain the implementation (const version) BEGIN_PROLOG

  NAME GetRequestImp

  DESCRIPTION Used to obtain the implementation object for this request

  RETURNS const pointer to the implementation object

  END_PROLOG

- GetRequestImp

  `virtual DsShGenRequestImp* GetRequestImp();`

  obtain the implementation BEGIN_PROLOG

  NAME GetRequestImp

  DESCRIPTION Used to obtain the implementation object for this request

  RETURNS pointer to the implementation object

  END_PROLOG

- operator =

  `DsClRequest& operator =(const DsClRequest& );`

  request to copy assignment operator NOT IMPLEMENTED

- saveOn

  `virtual void saveOn(ostream& os) const;`

  optional domain (i.e., which ESDTs) dumps human-readable text of an object BEGIN_PROLOG

  NAME saveOn

DESCRIPTION dumps human-readable text of an object

RETURNS nothing

END_PROLOG

- SetDomain

```
void SetDomain(RWTPtrOrderedVector<class GlUR>& domain);
```

set the domain for the request BEGIN_PROLOG

NAME SetDomain

DESCRIPTION set the domain for this request

RETURNS nothing

END_PROLOG

- SetPriority

```
void SetPriority(enum DsShGlobal::DsEShSciPriority pty);
```

set the priority for the request BEGIN_PROLOG

NAME SetPriority

DESCRIPTION sets the priority of the request

RETURNS nothing

END_PROLOG

- SetQuery

```
void SetQuery(DsClQuery* );
```

associate a query with the request

- Submit

```
virtual GlStatus Submit(DsClESDTReferenceCollector& erc,
RWTPtrOrderedVector<class GlUR>* domain);
```

submit to the DsClESDTReferenceCollector, with the given domain overloaded submit with domain

- Textify

```
virtual RWCString Textify() const;
```

convert this request to text form BEGIN_PROLOG

NAME Textify

DESCRIPTION Used to convert request into a text representation.

RETURNS the text representation

NOTES Until the final requirements are worked out, this doesn't do much.

END_PROLOG

4.3.7.2.13 DsClTypeInfo Class

Overview:

The following classes are defined in this file: DsClTypeInfo container class for product-type-level metadata

The following const variables are defined in this file: none

The following defines are defined in this file: _DsClTypeInfo_h_ make sure this isn't included more than once

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: DsClTypeInfo() default constructor DsClTypeInfo(const DsShTypeID& tid) construct w/o data DsClTypeInfo(const DsShTypeID& tid, construct with data GlParameterList& metadata, GlParameterList& queryableParameters, GlParameterList& scienceParameters) DsClTypeInfo(const DsClTypeInfo& rhs) copy constructor virtual ~DsClTypeInfo() destructor DsClTypeInfo& operator=(assignment operator const DsClTypeInfo& rhs) RWBoolean operator==(equality operator const DsClTypeInfo& rhs) const saveOn(ostream& os) const human-readable output GetTypeID() const return Type ID GetName() const return Type Name GetVersion() const return Type Version GetMetadata() const return type-level MD GetQueryableParameters() const return type-level QP GetScienceParameters() const return type-level SP SetTypeID(DsShTypeID* newType) set value for Type ID SetMetadata(const GlParameterList& metadata) put in metadata SetQueryableParameters(const GlParameterList& qp) put in QP data SetScienceParameters(const GlParameterList& sp) put in SP data operator<<(ostream& os, print object's data const DsClTypeInfo& ti)

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myMetadata`: `GlParameterList`

`myQueryableParameters`: `GlParameterList`

myScienceParameters: GlParameterList

myStatus: GlStatus

attributes

myTypeID: DsShTypeID*

Constructors and Destructor:

DsClTypeInfo(const DsClTypeInfo& rhs);

DsClTypeInfo(const DsShTypeID& tid);

default constructor

DsClTypeInfo(const DsShTypeID& tid, GlParameterList& metadata, GlParameterList& queryableParameters, GlParameterList& scienceParameters);

construct w/o data

DsClTypeInfo();

constructors

virtual ~DsClTypeInfo();

copy constructor destructor BEGIN_PROLOG

NAME ~DsClTypeInfo()

DESCRIPTION

Destructor, which removes the heap-allocated member data prior to the deallocation of this object.

RETURNS nothing, destructors can't return anything

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h GlParameterList.h

END_PROLOG

destructor

Operations:

- GetMetadata

```
virtual const GlParameterList& GetMetadata() const;
```

BEGIN_PROLOG

NAME GetMetadata() const

DESCRIPTION

This method provides access to this object's product-type metadata.

RETURNS const GlParameterList&, a reference to the type-level metadata of the product type which this object represents.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h GlParameterList.h

END_PROLOG

- GetName

```
virtual const RWCString& GetName() const;
```

BEGIN_PROLOG

NAME GetName() const

DESCRIPTION

This method provides access to this object's product-type name.

RETURNS const RWCString, the Name of the product type which this object represents.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h

END_PROLOG

- GetQueryableParameters

```
virtual const GlParameterList& GetQueryableParameters() const;
```

BEGIN_PROLOG

NAME GetQueryableParameters() const

DESCRIPTION

This method provides access to this object's product-type queryable parameters.

RETURNS const GlParameterList&, a reference to the type-level queryable parameters of the product type which this object represents.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h GlParameterList.h

END_PROLOG

- GetScienceParameters

```
virtual const GlParameterList& GetScienceParameters() const;
```

BEGIN_PROLOG

NAME GetScienceParameters() const

DESCRIPTION

This method provides access to this object's product-type science parameters.

RETURNS const GlParameterList&, a reference to the type-level science parameters of the product type which this object represents.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h GlParameterList.h

END_PROLOG

- GetTypeID

```
virtual const DsShTypeID& GetTypeID() const;
```

human-readable output gets

BEGIN_PROLOG

NAME GetTypeID() const

DESCRIPTION

This method provides access to this objects TypeID attribute.

RETURNS const DsShTypeID&, a reference to this object's TypeID attribute.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h

END_PROLOG

- GetVersion

```
virtual const RWCString& GetVersion() const;
```

BEGIN_PROLOG

NAME GetVersion() const

DESCRIPTION

This method provides access to this object's product-type version.

RETURNS const RWCString, the Version of the product type which this object represents.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h

END_PROLOG

- operator ==

```
int operator ==(const DsClTypeInfo& rhs) const;
```

BEGIN_PROLOG

NAME operator==(const DsClTypeInfo& rhs) const

DESCRIPTION

Equality operator, which indicates whether or not the provided object is content-equivalent to this object.

RETURNS RWBoolean, TRUE if the two objects are content-equivalent, FALSE otherwise

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h

END_PROLOG

equality operator

- operator =

```
DsClTypeInfo& operator =(const DsClTypeInfo& rhs);
```

operators operators

BEGIN_PROLOG

NAME operator=(const DsClTypeInfo& rhs)

DESCRIPTION

Assignment operator, which performs a deep assign of heap-allocated attributes so that this (new) object is autonomous.

RETURNS DsClTypeInfo&, the reference to this (the new object)

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h GlParameterList.h GlStatus.h

END_PROLOG

assignment operator

- saveOn

```
virtual void saveOn(ostream& os) const;
```

operations operations

BEGIN_PROLOG

NAME saveOn(ostream& os)

DESCRIPTION

This method produces human-readable output of this object's data on the provided stream. It is intended to be used with the overloaded operator<< defined in the DsClTypeInfo.h file.

RETURNS nothing, the output is placed on the provided stream.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h GlParameterList.h GlStatus.h

END_PROLOG

human-readable output

- SetMetadata

```
virtual GlStatus SetMetadata(const GlParameterList& metadata);
```

BEGIN_PROLOG

NAME SetMetadata(const GlParameterList& metadata)

DESCRIPTION

This method assigns the provided data to this object's Metadata attribute. A deep assign is performed so that this object remains autonomous.

RETURNS GlStatus object indicating success or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h GlParameterList.h GlStatus.h

END_PROLOG

- SetQueryableParameters

```
virtual GlStatus SetQueryableParameters(const GlParameterList&
queryableparms);
```

BEGIN_PROLOG

NAME SetQueryableParameters(const GlParameterList& queryableparms)

DESCRIPTION

This method assigns the provided data to this object's QueryableParameters attribute. A deep assign is performed so that this object remains autonomous.

RETURNS GlStatus object indicating success or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h GlParameterList.h GlStatus.h

END_PROLOG

- SetScienceParameters

```
virtual GlStatus SetScienceParameters(const GlParameterList&
scienceparms);
```

BEGIN_PROLOG

NAME SetScienceParameters(const GlParameterList& scienceparms)

DESCRIPTION

This method assigns the provided data to this object's ScienceParameters attribute. A deep assign is performed so that this object remains autonomous.

RETURNS GlStatus object indicating success or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h GlParameterList.h GlStatus.h

END_PROLOG

- SetTypeID

```
virtual GlStatus SetTypeID(DsShTypeID* newType);
```

sets

BEGIN_PROLOG

NAME SetTypeID(DsShTypeID* newType)

DESCRIPTION

This method assigns the provided value to this object's TypeID attribute.

RETURNS GlStatus object indicating success or failure of this operation.

GLOBALS none

FILES none

SYNOPSIS (optional) The following include files are required for this function: DsClTypeInfo.h DsShTypeID.h GlStatus.h

END_PROLOG

4.3.8 cn Class Category

4.3.8.1 Overview

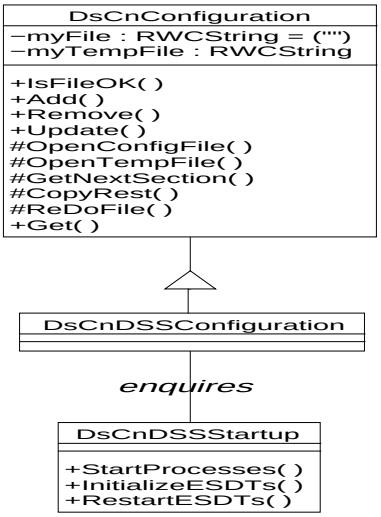


Figure 10 Main

4.3.8.2 cn Classes

4.3.8.2.1 DsCnConfiguration Class

Overview:

The following classes are defined in this file: DsCnConfiguration server configuration information

The following const variables are defined in this file: const char* DsCCnSectionStart = "[" begin section of file const char* DsCCnSectionEnd = "]" end section of file const char DsCCnAnySection = ' ' no section specified static const char section_delims[] = "[\\]\\[]"; delimiters between sections

The following defines are defined in this file: CONFIG_H ensure this file isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: DsCnConfiguration (const RWCString&); string filename DsCnConfiguration (); default protected ctor ~DsCnConfiguration(void); destructor IsFileOK(); check that file exists, readable Get (const RWCString&) const; get value of parameter Get (const RWCString&, const RWCString&) const; get from specific section Add (const RWCString&, const RWCString&, const RWCString&) const; add P=V to all sections between given sections Add (const RWCString&, const RWCString&); add P=V to section Remove (const RWCString&); remove P=V Remove (const RWCString&, const RWCString&); remove P=V from section Update (const RWCString&, const RWCString&); replace P=V Update (const RWCString&, const RWCString&, const RWCString&); replace P=V in section OpenConfigFile(ifstream&) const ; open object's file RWBoolean OpenTempFile(open temporary file ofstream& inFile) const ; stream for writing GetNextSection(ifstream&, ostream* = NULL) const ; get next section value, position at next line RWCString GetParameter(RWCString inParameter, parameter to find ifstream& inFile, file from which to read ofstream* outFile = NULL) ; [opt] file to write void CopyRest(ifstream& inFile, original config file ofstream& outFile) const ; modified config file ReDoFile(rewrite file ifstream& configFile, original config file ofstream& tempFile) const modified config file GetSectionToken(const RWCString&, RWCString&) const; get next section name SetFileName(const RWCString &) configuration filename

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myFile: RWCString`

`myTempFile: RWCString`

Constructors and Destructor:

`DsCnConfiguration(const RWCString& fname);`

member initialization list instead of assignment

`DsCnConfiguration();`

location for output

```
virtual  ~DsCnConfiguration(void );
```

filename BEGIN_PROLOG

NAME ~DsCnConfiguration()

DESCRIPTION The destructor removes objects of this type. There is no cleanup to do.

RETURNS nothing, it's a destructor

GLOBALS none

FILES none

END_PROLOG

Operations:

- Add

```
int Add(const RWCString& inSection, const RWCString&
inParameter, const RWCString& inValue);
```

parameter name BEGIN_PROLOG

NAME Add(const RWCString& inSection, // section to which to add const RWCString& inParameter, // parameter name const RWCString& inValue) // value

DESCRIPTION

This operation adds a new (previously unrecorded) parameter/value line to the specified section. There are no checks to see if the value already exists; it is possible to have more than one line for a given parameter within a given section. The routine works by reading from the config file and writing to a temporary file. Each section header line is checked to see if it is the desired one. All other lines are copied straight to output. If the section header line is the correct one, then a new P=V line is inserted into the file immediately after the section header line.

RETURNS RWBoolean, FALSE if any error is encountered, TRUE otherwise

GLOBALS none

FILES The configuration file named in attribute myFile, and a temporary file for the modified configuration file (which will later be renamed to be the permanent file).

END_PROLOG

- Add

```
int Add(const RWCString& inParameter, const RWCString& inValue);
```

value BEGIN_PROLOG

NAME Add(const RWCString& inParameter, // parameter name const RWCString& inValue) // value

DESCRIPTION

This operation provides a convenient interface to Add the new parameter/value without specifying a section name. This operation just calls the Add(section, parameter, value) function with DsCCnAnySection as the section of interest.

RETURNS RWBoolean, FALSE if any error is encountered, TRUE otherwise

GLOBALS none

FILES none

END_PROLOG

- CopyRest

```
void CopyRest(ifstream& inFile, ofstream& outFile) const;
```

[opt] file to write BEGIN_PROLOG

NAME RWBoolean CopyRest(ifstream& inFile, // original config file ofstream& outFile) const ; // modified config file

DESCRIPTION

This operation copies the remaining data from inFile to outFile.

RETURNS nothing

GLOBALS none

FILES The original configuration data file, and the temp file created and written to by the modifying operations of this object.

END_PROLOG

- GetNextSection

```
RWCString GetNextSection(ifstream& inFile, ofstream* outFile)
const;
```

stream to use for writing BEGIN_PROLOG

NAME GetNextSection ifstream& inFile, // file from which to read ofstream* outFile) const // [opt] file to write

DESCRIPTION

This operation returns the value of the next section header name, and positions the file pointer at the line following the section header (i.e., at the first P=V line of the given section). Optionally, if an output file has been specified (i.e., for operations which modify the file), the lines which are passed are copied to the output file.

RETURNS RWCString, the value of the section header name

GLOBALS none

FILES The specified configuration data file, and an optional (temporary) output file

END_PROLOG

- GetParameter

```
RWCString GetParameter(const RWCString& inParameter, ifstream&
inFile, ofstream* outFile) const;
```

[opt] file to write BEGIN_PROLOG

NAME GetParameter RWCString& inParameter, // parameter to find ifstream& inFile, // file from which to read ofstream* outFile) const // [opt] file to write

DESCRIPTION

This operation returns the value of the specified parameter, and positions the file pointer at the line following the given parameter. If a blank line is encountered, then the end of the section has been reached without having found a parameter, so a value of "" is returned, and the file pointer is pointing to the next section header line. Optionally, if an output file has been specfied (i.e., for operations which modify the configuration file), the lines which are passed are copied to the output file.

RETURNS RWCString, the value of the specified parameter, or NULL if the parameter was not found in this section.

GLOBALS none

FILES The specified configuration data file, and an optional (temporary) output file

END_PROLOG

- GetSectionToken

`unsigned int GetSectionToken(const RWCString& buffer, RWCString& token) const;`

modified config file BEGIN_PROLOG

NAME GetSectionToken (const RWCString& buffer, // where to look RWCString& token) const // where to put results

DESCRIPTION

This operation reads the provided line and returns the section name if the line is a section header line, and returns an empty string if the line is either a comment line or a non-section header line. The check for comment lines comes first so that section header lines can be commented out.

RETURNS The section name in parameter 2, and size_t, length of the section header name

GLOBALS none

FILES none

END_PROLOG

- Get

`RWCString Get(const RWCString& inParameter) const;`

file readable BEGIN_PROLOG

NAME Get(const RWCString& inParameter) const // parameter to find

DESCRIPTION

This operation provides a convenient interface to search for the specified parameter without specifying a section name. This operation just calls the Get(section, parameter) function with DsCCnAnySection as the section of interest.

RETURNS RWCString, the value of the desired parameter, or none, if the parameter is not there.

GLOBALS none

FILES none

END_PROLOG

- Get

```
RWCString Get(const RWCString& inSection, const RWCString&
inParameter) const;
```

parameter name Begin_PROLOG

NAME Get(const RWCString& inSection, // section to search const RWCString& inParameter) const // parameter to find

DESCRIPTION

This operation finds and returns the value for the given parameter. If inSection is DsCCnAnySection, then the value of the first parameter matching the specified parameter is taken. If inSection is the name of a section, then only the parameter of the correct name from the specified section will be found.

RETURNS RWCString, the value of the desired parameter, or none, if the parameter is not there.

GLOBALS none

FILES The configuration file named in attribute myFile.

END_PROLOG

- IsFileOK

```
virtual int IsFileOK();
```

destructor BEGIN_PROLOG

NAME IsFileOK ()

DESCRIPTION

This operation checks that this configuration file exists and is readable

RETURNS RWBoolean, FALSE if the file can't be opened for read, TRUE otherwise

GLOBALS none

FILES The specified configuration data file

END_PROLOG

- OpenConfigFile

```
int OpenConfigFile(ifstream& inFile) const;
```

BEGIN_PROLOG

NAME OpenConfigFile(ifstream& inFile) const

DESCRIPTION

This operation opens the configuration file identified in this object's attribute myFile for read access.

RETURNS RWBoolean, FALSE if the file can't be opened for read, TRUE otherwise

GLOBALS none

FILES The specified configuration data file

END_PROLOG

- OpenTempFile

```
int OpenTempFile(ofstream& outFile);
```

stream to use for reading BEGIN_PROLOG

NAME OpenTempFile(ofstream& outFile)

DESCRIPTION

This operation opens a temporary file for output of modified config file.

RETURNS RWBoolean, FALSE if the file can't be opened for read, TRUE otherwise

GLOBALS none

FILES The system-provided temporary file

END_PROLOG

- ReDoFile

```
int ReDoFile(ifstream& configFile, ofstream& tempFile) const;
```

modified config file BEGIN_PROLOG

NAME ReDoFile() const

DESCRIPTION

This operation rewrites the file specified by the myFile attribute after new data has been recorded. This is done by renaming the temp file into which the changes were recorded to the original filename.

RETURNS RWBoolean, FALSE if the original file can't be unlinked or the temp file can't be renamed, TRUE otherwise

GLOBALS none

FILES The original configuration data file, and the temp file created and written to by the modifying operations of this object.

END_PROLOG

- Remove

```
int Remove(const RWCString& inSection, const RWCString&
inParameter);
```

parameter name BEGIN_PROLOG

NAME Remove(const RWCString& inSection, // section to change const RWCString& parameter) // parameter name

DESCRIPTION

This operation removes a specified parameter/value line from the specified section. The routine works by reading from the config file and writing to a temporary file. Each section header line is checked to see if it is the specified one. All section header lines are written back to the output. All non-header lines that are in the specified section are checked against the specified parameter (all non-header lines in non-interesting sections are copied straight to ouput), and if the parameter matches, then the line is NOT written to output.

RETURNS RWBoolean, FALSE if any error is encountered, TRUE otherwise

GLOBALS none

FILES The configuration file named in attribute myFile, and a temporary file for the modified configuration file (which will later be renamed to be the permanent file).

END_PROLOG

- Remove

```
int Remove(const RWCString& inParameter);
```

value BEGIN_PROLOG

NAME Remove(const RWCString& parameter) // parameter name

DESCRIPTION

This operation provides a convenient interface to Remove the specified parameter/value without specifying a section name. This operation just calls the Remove(section, parameter) function with DsCCnAnySection as the section of interest.

RETURNS RWBoolean, FALSE if any error is encountered, TRUE otherwise

GLOBALS none

FILES none

END_PROLOG

- SetFileName

```
virtual void SetFileName(const RWCString& fname);
```

default constructor BEGIN_PROLOG

NAME SetFileName()

DESCRIPTION

Set the filename of this configuration object to the given string

RETURNS nothing

GLOBALS none

FILES none

NOTES Use of this constructor requires a call to SetFileName before anything useful can be done END_PROLOG

- Update

```
int Update(const RWCString& inSection, const RWCString&
inParameter, const RWCString& inValue);
```

value BEGIN_PROLOG

NAME Update(const RWCString& inSection, // section to update const RWCString& inParameter, // parameter name const RWCString& inValue) // new value

DESCRIPTION

This operation updates an existing parameter/value line in the specified section. The routine works by reading from the config file and writing to a temporary file. Each section header line is checked to see if it is the desired section (and the section header line is written to output). If a non-header section line is written, the parameter name is extracted. If the section and parameter are the one specified, then the new value is written to output. If the section and parameter are not the one specified, the then line is copied to output as it is.

RETURNS RWBoolean, FALSE if any error is encountered, TRUE otherwise

GLOBALS none

FILES The configuration file named in attribute myFile, and a temporary file for the modified configuration file (which will later be renamed to be the permanent file).

END_PROLOG

- Update

```
int Update(const RWCString& inParameter, const RWCString&
inValue);
```

parameter name BEGIN_PROLOG

NAME Update(const RWCString& inParameter, // parameter name const RWCString& inValue) // new value

DESCRIPTION

This operation provides a convenient interface to Update the specified parameter/value without specifying a section name. This operation just calls the Update(section, parameter, value) function with DsCCnAnySection as the section of interest.

RETURNS RWBoolean, FALSE if any error is encountered, TRUE otherwise

GLOBALS none

FILES none

END_PROLOG

4.3.8.2.2 DsCnDSSConfiguration Class

Overview:

Distributed Object: No This is a datserver configuration startup class that starts processes as defined in a configuration file.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCnConfiguration

Attributes:

Constructors and Destructor:

Operations:

- GetProcessesToStart

  ```
  GetProcessesToStart();
  ```

  Uses the base configuration class to read processes from a file in order to initiate processes in that file.

4.3.8.2.3 DsCnDSSStartup Class

Overview:

Distributed Object: No This starts up a all processes for a particular dataserver by enquiring a particular configuration file and initializing/restarting ESDT's. This starts up all processes for a particular dataserver by enquiring a particular configuration file and initializing/restarting ESDTs.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsCnDSSStartup();
```

Constructs a DsCnDSSStartup class.

```
~DsCnDSSStartup();
```

Operations:

- InitializeESDTs

  ```
  RWBoolean InitializeESDTs();
  ```

  Creates descriptors for each type in configuration.

- RestartESDTs

  ```
  RWBoolean RestartESDTs();
  ```

  If something should happen to the advertisement server, then redo the initialize ESDTs and "re-advertise" them.

- StartProcesses

  ```
  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.

4.3.9 co Class Category

4.3.9.1 Overview

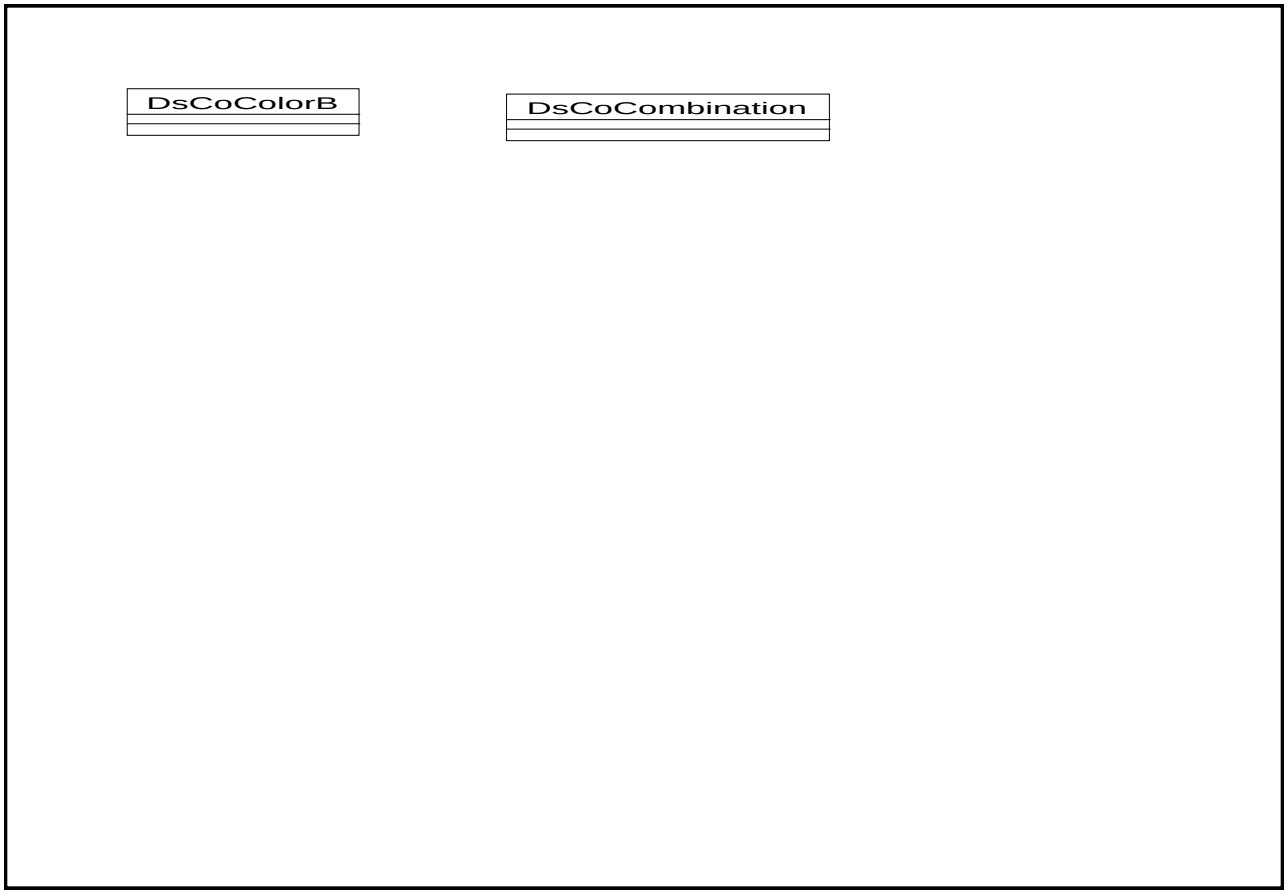


Figure11Main

4.3.9.2 co Classes

4.3.9.2.1 DsCoColorB Class

Overview:

Distributed Object: No This class represents products generated from the Color Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsCoColorB();

~DsColorB();
```

Operations:

4.3.9.2.2 DsCoCombination Class

Overview:

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.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

Operations:

4.3.10 cs Class Category

4.3.10.1 Overview

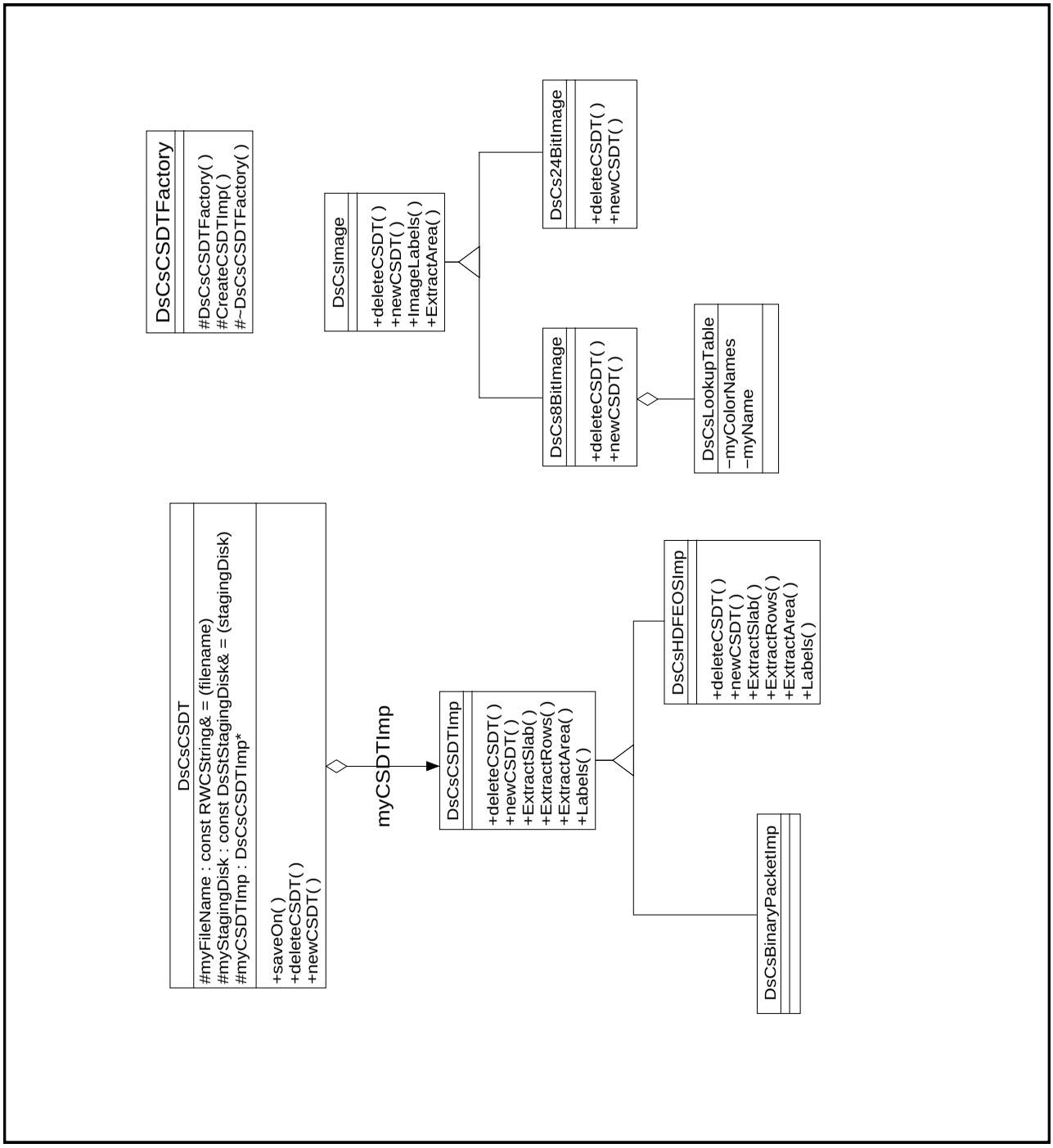


Figure 12 DsCs1

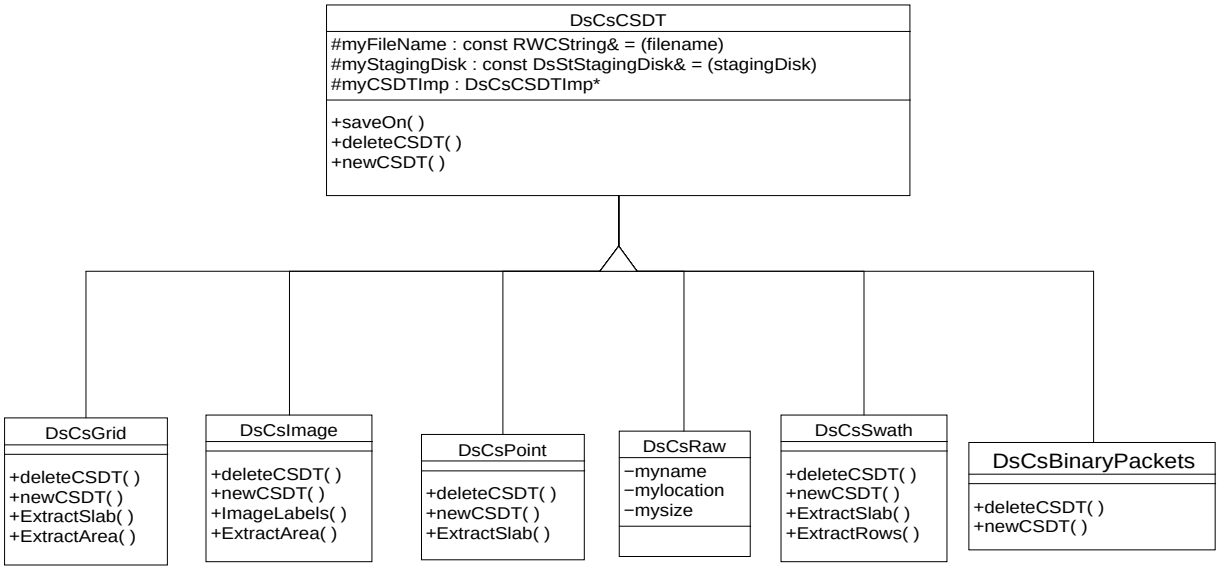


Figure 13 DsCs2

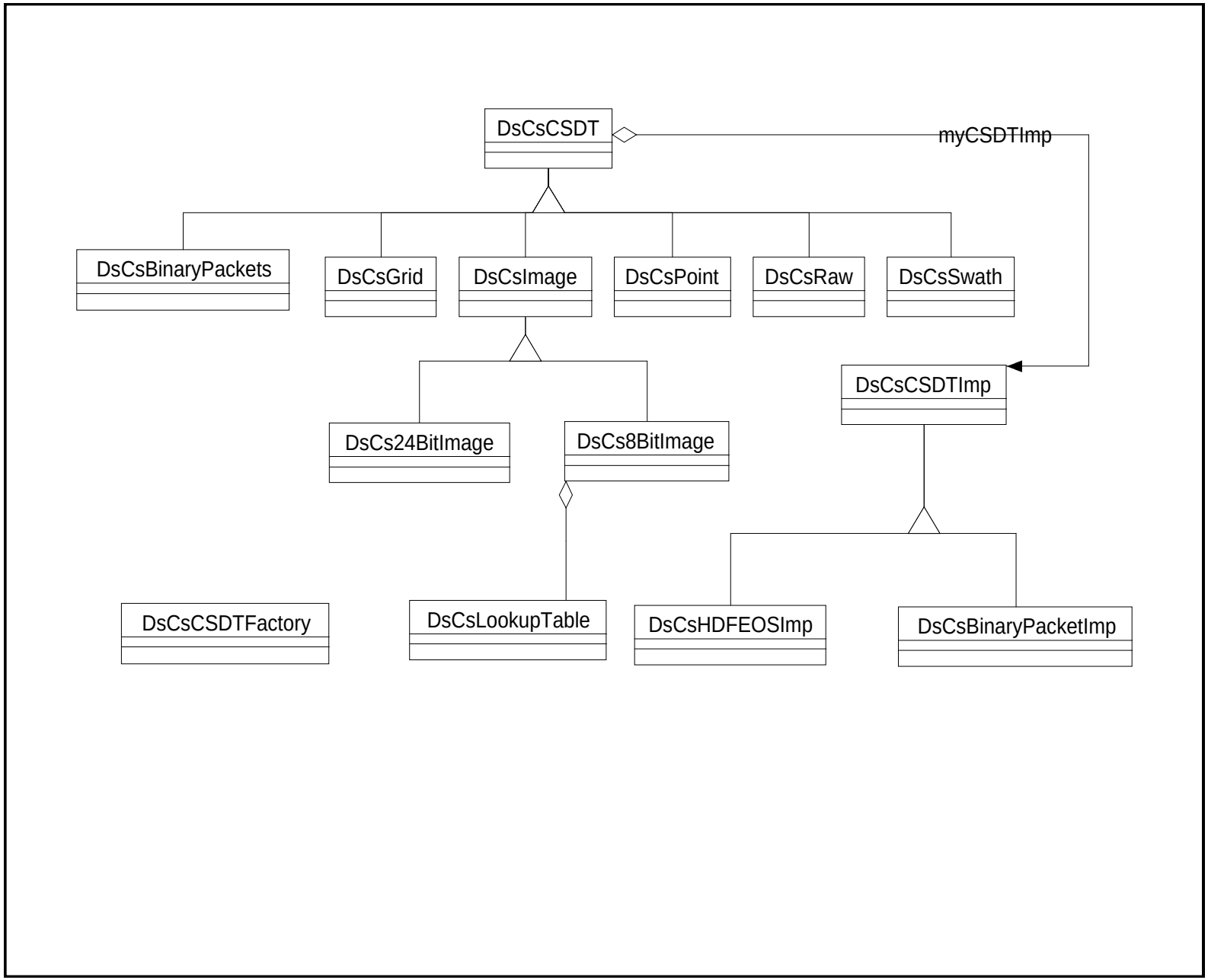


Figure 14 Main

4.3.10.2 cs Classes

4.3.10.3 DsCs24BitImage Class

Overview:

The following classes are defined in this file: DsCs24Bit24BitImageH

The following const variables are defined in this file: none

The following defines are defined in this file: DsCs24BitImageH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsImage

Attributes:

Constructors and Destructor:

```
DsCs24BitImage(const RWCString& CSDTImpType, const
DsStStagingDisk& stagingDisk, const RWCString& filename);
```

constructor

```
virtual  ~DsCs24BitImage();
```

BEGIN_PROLOG

NAME DsCs24BitImage

DESCRIPTION Default destructor

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

`virtual GlStatus deleteCSDT();`

Delete CSDT - different from destructor This will delete the file from disk aswell
BEGIN_PROLOG

NAME deleteCSDT

DESCRIPTION Destroys <this> CSDT AND deletes associated file

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- GetFileStructure

`virtual void GetFileStructure(ostream& os) const;`

makes sure the derived destructors get called

Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

Output file structure Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

BEGIN_PROLOG

NAME GetFileStructure

DESCRIPTION Places file structure description on ouput stream.

INPUTS none

OUTPUTS ostream

RETURNS none

END_PROLOG

- GetInterleavingType

`virtual RWCString GetInterleavingType() const;`

24BitImage specific methods - File format dependent methods forwarded to implementor

Get the type of interleaving used Image specific methods - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetInterleavingType

DESCRIPTION get type of interleaving used.

INPUTS none

OUTPUTS none

RETURNS RWCString
END_PROLOG

- newCSDT

```
virtual GlStatus newCSDT(const RWCString& filename);
```

Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG
NAME newCSDT
DESCRIPTION Create a new CSDT with a new (empty) point file
INPUTS RWCString // filename to call new point file
OUTPUTS none
RETURNS GlStatus
END_PROLOG

4.3.10.3.1 DsCs8BitImage Class

Overview:

The following classes are defined in this file: DsCs8BitImageH

The following const variables are defined in this file: none

The following defines are defined in this file: DsCs8BitImageH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

DIstributed Object: No

Export Control: Public

Inheritance Relationships:

Inherits from DsCsImage

Attributes:

Constructors and Destructor:

```
DsCs8BitImage(const RWCString& CSDTImpType, const
DsStStagingDisk& stagingDisk, const RWCString& filename);
```

constructor

```
virtual  ~DsCs8BitImage();
```

BEGIN_PROLOG

NAME DsCs8BitImage

DESCRIPTION Default destructor

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

```
virtual GlStatus deleteCSDT();
```

Delete CSDT - different from destructor This will delete the file from disk aswell
BEGIN_PROLOG

NAME deleteCSDT

DESCRIPTION Destroys <this> CSDT AND deletes associated file

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- GetFileStructure

```
virtual void GetFileStructure(ostream& os) const;
```

makes sure the derived destructors get called

Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

Output file structure Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

BEGIN_PROLOG

NAME GetFileStructure

DESCRIPTION Places file structure description on ouput stream.

INPUTS none

OUTPUTS ostream

RETURNS none

END_PROLOG

- GetLookUptable

  `virtual void GetLookUptable() const;`

  8BitImage specific methods - File format dependent methods forwarded to implementor

  Get Look Up Table Image specific methods - File format dependent methods forwarded to implementor

  BEGIN_PROLOG

  NAME GetLookUptable

  DESCRIPTION get look up table used by <this> 8 bit image.

  INPUTS none

  OUTPUTS none

  RETURNS void

  END_PROLOG

- newCSDT

  `virtual GlStatus newCSDT(const RWCString& filename);`

  Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG

  NAME newCSDT

  DESCRIPTION Create a new CSDT with a new (empty) point file

  INPUTS RWCString // filename to call new point file

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

4.3.10.3.2 DsCsBinaryPacketImp Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsCSDTImp

Attributes:

Constructors and Destructor:

Operations:

4.3.10.3.3 DsCsBinaryPackets Class

Overview:

The following classes are defined in this file: DsCsBinaryPackets N.B. This class used to be called DsCsRaw.

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsBinaryPacketsH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No Class definition

Provides statement of binary packets "raw" service availability

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsCSDT

Attributes:

Constructors and Destructor:

```
DsCsBinaryPackets(const RWCString& CSDTImpType, const
DsStStagingDisk& stagingDisk, const RWCString& filename);
```

```
~DsCsBinaryPackets();
```

BEGIN_PROLOG

NAME ~DsCsBinaryPackets

DESCRIPTION Destructor for DsCsBinaryPackets CSDT. This function is declared virtual to ensure derived class destructors are called.

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

  ```
  virtual GlStatus deleteCSDT();
  ```

  Delete CSDT - different from destructor This will also delete the file from disk BEGIN_PROLOG

  NAME deleteCSDT

  DESCRIPTION Destroys <this> CSDT AND deletes associated file

  INPUTS none

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

- GetCSDTImpVersionNo

  ```
  virtual RWCString GetCSDTImpVersionNo() const;
  ```

  Get version number of implementation BEGIN_PROLOG

  NAME GetCSDTImpVersionNo

  DESCRIPTION Returns version number of CSDT implementation.

  INPUTS none

  OUTPUTS none

  RETURNS RWCString

  END_PROLOG

- GetCSDTVersionNo

  ```
  virtual RWCString GetCSDTVersionNo() const;
  ```

  Get version number of CSDT Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

  BEGIN_PROLOG

  NAME GetCSDTVersionNo

  DESCRIPTION Returns version number of <this> CSDT.

  INPUTS none

  OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetFileStructure

  `virtual void GetFileStructure(ostream& os) const;`

  Output file structure Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

  BEGIN_PROLOG

  NAME GetFileStructure

  DESCRIPTION Places file structure description on output stream.

  INPUTS none

  OUTPUTS ostream

  RETURNS none

  END_PROLOG

- newCSDT

  `virtual GlStatus newCSDT(const RWCString& filename);`

  Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG

  NAME newCSDT

  DESCRIPTION Create a new CSDT with a new (empty) binary packets file

  INPUTS RWCString // filename to call new grid file

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

4.3.10.3.4 DsCsCSDTFactory Class

Overview:

The following classes are defined in this file: DsCsCSDTFactory

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsCSDTFactory

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsCsCSDTFactory();
```

BEGIN_PROLOG

NAME Constructor

DESCRIPTION Default constructor for factory

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

```
~DsCsCSDTFactory();
```

Operations:

- CreateCSDTImp

```
virtual DsCsCSDTImp* CreateCSDTImp(const RWCString& CSDTImpType,
const RWCString& filename);
```

4.3.10.3.5 DsCsCSDTImp Class

Overview:

The following classes are defined in this file: DsCsCSDTImp

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsCSDTImpH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsCsCSDTImp();
```

BEGIN_PROLOG
NAME DsCsCSDTImp
DESCRIPTION Default Constructor for CSDT Implementor
INPUTS none
OUTPUTS none
RETURNS none
END_PROLOG

```
DsCsCSDTImp(const RWCString& fileName);
```

BEGIN_PROLOG
NAME DsCsCSDTImp
DESCRIPTION Constructor for CSDT Implementor
INPUTS RWCString& - file to use for construction
OUTPUTS none
RETURNS none
END_PROLOG

```
virtual  ~DsCsCSDTImp();
```

BEGIN_PROLOG
NAME ~DsCsCSDTImp
DESCRIPTION Default destructor for CSDT Implementor
INPUTS none
OUTPUTS none
RETURNS none
END_PROLOG

Operations:

- deleteCSDT

```
virtual GlStatus deleteCSDT() = 0;
```

Delete CSDT - different from destructor This will also delete the file from disk

- ExtractArea

```
virtual GlStatus ExtractArea(const long TLlatitude, const long TLlongitude, const long BRlatitude, const long BRlongitude, const long height, const long longitudeStride, const long latitudeStride) = 0;
```

Subset or subsample by user defined rectangular box.

- ExtractRows

```
virtual GlStatus ExtractRows(const int startRowNum, const int endRowNum, int actualEndRowNum, const int stride) = 0;
```

Subset or Subsample by Complete Row, using start row and stop row, or start row and no. rows forward.

- ExtractSlab

```
virtual GlStatus ExtractSlab(const GlTimeP& starttime, const GlTimeP& stoptime, const int stride) = 0;
```

Subset or Subsample by time, using start time and stop time

- ExtractSlab

```
virtual GlStatus ExtractSlab(const RWCString& parameterName, const int stride) = 0;
```

Subset or Subsample by parameter name

- ExtractSlab

```
virtual GlStatus ExtractSlab(const GlTimeP& orbittime, const long duration, const int stride) = 0;
```

Subset or Subsample by time, using start time and duration (in seconds)

- GetCompressionName

```
virtual RWCString GetCompressionName() const = 0;
```

- GetCSDTImpVersionNo

```
virtual RWCString GetCSDTImpVersionNo() const = 0;
```

All Service methods are abstract - must be format specific implementations

version number of implementor

- GetDimensions

```
virtual RWCString GetDimensions() const = 0;
```

- GetFileStructure

```
virtual void GetFileStructure(ostream& os) const = 0;
```

Output file structure

- GetGeophysicalParameters

```
virtual RWTPtrHashSet<class RWCString>
GetGeophysicalParameters() const = 0;
```

- GetInterleavingType

```
virtual RWCString GetInterleavingType() const = 0;
```

Return interleaving used

- GetLookUptable

```
virtual void GetLookUptable() const = 0;
```

Return Look up Table in use

- GetOrbitRepresentation

```
virtual RWCString GetOrbitRepresentation() const = 0;
```

- GetRecordType

```
virtual RWCString GetRecordType() const = 0;
```

Return record type for point data

- GetResolution

```
virtual long GetResolution() const = 0;
```

- GetTransformationName

```
virtual RWCString GetTransformationName() const = 0;
```

name of geometric transformation used

- Labels

  ```
  virtual RWTPtrHashSet<class RWCString> Labels() const = 0;
  ```

  Return record type for image data

- newCSDT

  ```
  virtual GlStatus newCSDT(const RWCString& filename) = 0;
  ```

  Create CSDT - different from constructor This will create a new file on disk

- SetLookUptable

  ```
  virtual GlStatus SetLookUptable() const = 0;
  ```

  Set Look up Table to use

4.3.10.3.6 DsCsCSDT Class

Overview:

The following classes are defined in this file: DsCsCSDT

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsCSDTH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following classes are defined in this file: DsCsCSDTFactory

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsCSDTFactory

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following classes are defined in this file: DsCsCSDTImp

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsCSDTImpH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No Class definition Provides basic CSDT functionality

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myCSDTImp: DsCsCSDTImp*
```

```
myFileName: const RWCString&
```

```
myStagingDisk: const DsStStagingDisk&
```

Constructors and Destructor:

```
DsCsCSDT(const RWCString& CSDTImpType, const DsStStagingDisk&
stagingDisk, const RWCString& filename);
```

Generic CSDT methods implemented by this base class

constructor

```
virtual  ~DsCsCSDT();
```

destructor BEGIN_PROLOG

NAME ~DsCsCSDT

DESCRIPTION Destructor for all CSDTs

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

```
virtual GlStatus deleteCSDT() = 0;
```

Delete CSDT - different from destructor This will also delete the file from disk

- GetCSDTImpVersionNo

```
virtual RWCString GetCSDTImpVersionNo() const = 0;
```

Get version number of CSDT implementation

- GetCSDTVersionNo

  ```
  virtual RWCString GetCSDTVersionNo() const = 0;
  ```

  Get version number of CSDT

- GetFileName

  ```
  const RWCString& GetFileName();
  ```

  Get filename CSDT represents BEGIN_PROLOG
  NAME GetFileName
  DESCRIPTION Returns the filename that this CSDT represents
  INPUTS none
  OUTPUTS none
  RETURNS RWCString
  END_PROLOG

- GetFileSize

  ```
  virtual long GetFileSize() const;
  ```

  O: Output stream GetSize of file on disk. BEGIN_PROLOG
  NAME GetFileSize
  DESCRIPTION Return the size of the file (Kb) represented by <this> CSDT
  INPUTS none
  OUTPUTS none
  RETURNS none
  END_PROLOG

- GetFileStructure

  ```
  virtual void GetFileStructure(ostream& os) const = 0;
  ```

  Generic methods that must be defined by CSDT specializations
  Output file structure

- newCSDT

  ```
  virtual GlStatus newCSDT(const RWCString& filename) = 0;
  ```

  Create CSDT - different from constructor This will create a new file on disk

- operator ==

  ```
  virtual int operator ==(const DsCsCSDT& rhs);
  ```

  Equality operator BEGIN_PROLOG
  NAME GetFileSize

DESCRIPTION Equality operator
INPUTS DsCsCSDT
OUTPUTS none
RETURNS RWBoolean
END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

Comparison overload Save internals of CSDT on output stream BEGIN_PROLOG
NAME saveOn
DESCRIPTION Output contents of CSDT
INPUTS ostream
OUTPUTS none
RETURNS none
END_PROLOG

4.3.10.3.7 DsCsGrid Class

Overview:

The following classes are defined in this file: DsCsGridH

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsGridH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No Class definition Provides basic CSDT functionality

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsCSDT

Attributes:

Constructors and Destructor:

```
DsCsGrid(const RWCString& CSDTImpType, const DsStStagingDisk&
stagingDisk, const RWCString& filename);
```

constructor

```
virtual  ~DsCsGrid();
```

BEGIN_PROLOG

NAME DsCsGrid

DESCRIPTION Default destructor

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

```
virtual GlStatus deleteCSDT();
```

Delete CSDT - different from destructor This will also delete the file from disk
BEGIN_PROLOG

NAME deleteCSDT

DESCRIPTION Destroys <this> CSDT AND deletes associated file

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractArea

```
virtual GlStatus ExtractArea(const long TLlatitude, const long
TLlongitude, const long BRlatitude, const long BRlongitude, const
long height, const long longitudeStride, const long
latitudeStride);
```

Subset or Subsample by rectangular coords. BEGIN_PROLOG

NAME ExtractArea

DESCRIPTION Extract area of data from grid by spatial location.

INPUTS const long TLlatitude, const long TLlongitude, const long BRlatitude, const long BRlongitude, const long height, const long longitudeStride, const long latitudeStride

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractSlab

```
virtual GlStatus ExtractSlab(const GlTimeP& starttime, const GlTimeP& stoptime, const int stride);
```

Subset or Subsample by time, using start time and stop time BEGIN_PROLOG

NAME ExtractSlab

DESCRIPTION Extract 'slab' of data from grid by time.

INPUTS GlTimeP& - start time GlTimeP& - end time int - stride along grid

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractSlab

```
virtual GlStatus ExtractSlab(const RWCString& parameterName, const int stride);
```

Subset or Subsample by parameter name BEGIN_PROLOG

NAME ExtractSlab

DESCRIPTION Extract 'slab' of data from grid by parameter.

INPUTS RWCString& - name of param to extract int - stride along grid

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- GetCompressionName

```
virtual RWCString GetCompressionName() const;
```

BEGIN_PROLOG

NAME GetCompressionName

DESCRIPTION Return name of compression algorithm used on file

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetCSDTImpVersionNo

```
virtual RWCString GetCSDTImpVersionNo() const;
```

Get version number of Grid CSDT Implementation BEGIN_PROLOG

NAME GetCSDTImpVersionNo

DESCRIPTION Returns version number of grid CSDT implementation.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetCSDTVersionNo

```
virtual RWCString GetCSDTVersionNo() const;
```

makes sure the derived destructors get called

Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

Get version number of Grid CSDT Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

BEGIN_PROLOG

NAME GetCSDTVersionNo

DESCRIPTION Returns version number of <this> grid CSDT.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetDimensions

```
virtual RWCString GetDimensions() const;
```

BEGIN_PROLOG

NAME GetDimensions

DESCRIPTION Return array dimensions (in meters) of science data

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetFileStructure

```
virtual void GetFileStructure(ostream& os) const;
```

Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

Output file structure Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetFileStructure

DESCRIPTION Places file structure description on output stream.

INPUTS none

OUTPUTS ostream

RETURNS none

END_PROLOG

- GetGeophysicalParameters

```
virtual RWTPtrHashSet<class RWCString>
GetGeophysicalParameters() const;
```

BEGIN_PROLOG

NAME GetGeophysicalParameters

DESCRIPTION Return searchable geophysical parameters

INPUTS none

OUTPUTS none

RETURNS RWTPtrHashSet<RWCString>

END_PROLOG

- GetResolution

```
virtual long GetResolution() const;
```

BEGIN_PROLOG

NAME GetResolution

DESCRIPTION Return resolution (in meters) of science data

INPUTS none

OUTPUTS none

RETURNS long

END_PROLOG

- GetTransformationName

```
virtual RWCString GetTransformationName() const;
```

Grid specific methods - File format dependent methods forwarded to implementor

Grid specific methods - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetTransformationName

DESCRIPTION Return grid Transformation name to client

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- newCSDT

```
virtual GlStatus newCSDT(const RWCString& filename);
```

Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG

NAME newCSDT

DESCRIPTION Create a new CSDT with a new (empty) grid file

INPUTS RWCString // filename to call new grid file

OUTPUTS none

RETURNS GlStatus

END_PROLOG

4.3.10.3.8 DsCsHDFEOSImp Class

Overview:

The following classes are defined in this file: DsCsHDFEOSImp Swath HDF_EOS Implementation class

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsHDFEOSImpH ensures file isn't read twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No Provides swath functionality using HDF_EOS

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsCSDTImp

Attributes:

Constructors and Destructor:

```
DsCsHDFEOSImp(const RWCString& FileName);
```

BEGIN_PROLOG

NAME DsCsHDFEOSImp

DESCRIPTION Constructor for DsCsHDFEOSImp CSDT.

INPUTS RWCString&

OUTPUTS none

RETURNS none

END_PROLOG

```
~DsCsHDFEOSImp();
```

BEGIN_PROLOG

NAME ~DsCsHDFEOSImp

DESCRIPTION Destructor for DsCsHDFEOSImp CSDT. This function is declared virtual to ensure derived class destructors are called.

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

```
virtual GlStatus deleteCSDT();
```

Delete CSDT - different from destructor This will also delete the file from disk
BEGIN_PROLOG

NAME deleteCSDT

DESCRIPTION Delete <this> CSDT AND the associated file.

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractArea

```
virtual GlStatus ExtractArea(const long TLlatitude, const long
TLlongitude, const long BRlatitude, const long BRlongitude, const
long height, const long longitudeStride, const long
latitudeStride);
```

Subset or subsample by user defined rectangular box. BEGIN_PROLOG

NAME ExtractArea

DESCRIPTION Extract data based on spatial constraint defined as rectangular bounding box and a fixed height.

INPUTS const long TLlatitude - top left latitude const long TLlongitude - top left longitude const long BRlatitude - bottom right latitude const long BRlongitude - bottom right longitude const long height - fixed height const long longitudeStride - stride along longitude const long latitudeStride - stride along latitude

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractRows

```
virtual GlStatus ExtractRows(const int startRowNum, const int
endRowNum, int actualEndRowNum, const int stride);
```

Subset or Subsample by Complete Row, using start row and stop row, or start row and no. rows forward. BEGIN_PROLOG

NAME ExtractRows

DESCRIPTION Extract complete rows of data using row start & end number

INPUTS int - start row numbers int - end row number/number of rows forward RWBoolean - TRUE = end row number / FALSE = num rows forward int - stride (subsampling)

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractSlab

```
virtual GlStatus ExtractSlab(const GlTimeP& starttime, const
GlTimeP& stoptime, const int stride);
```

Subset or Subsample by time, using start time and stop time BEGIN_PROLOG

NAME ExtractSlab

DESCRIPTION Extract slab of data using time as constraint.

INPUTS GlTimeP& - start orbit time GlTimeP& - stop time int - stride (subsampling)

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractSlab

```
virtual GlStatus ExtractSlab(const RWCString& parameterName,
const int stride);
```

Subset or Subsample by parameter name BEGIN_PROLOG

NAME ExtractSlab

DESCRIPTION Extract slab of data using parameter name as constraint.

INPUTS RWCString& - name of parameter int - stride (subsampling)

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractSlab

```
virtual GlStatus ExtractSlab(const GlTimeP& orbittime, const
long duration, const int stride);
```

Subset or Subsample by time, using start time and duration (in seconds) BEGIN_PROLOG

NAME ExtractSlab

DESCRIPTION Extract slab of data using time as constraint.

INPUTS GlTimeP& - start orbit time long - duration from start int - stride (subsampling)

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- GetCompressionName

```
virtual RWCString GetCompressionName() const;
```

BEGIN_PROLOG

NAME GetCompressionName

DESCRIPTION The name of orbit algorithm used for this CSDT.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetCSDTImpVersionNo

```
virtual RWCString GetCSDTImpVersionNo() const;
```

version number of implememtor BEGIN_PROLOG

NAME GetCSDTImpVersionNo

DESCRIPTION Get version number for this HDFEOS implementation.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetDimensions

```
virtual RWCString GetDimensions() const;
```

BEGIN_PROLOG

NAME GetDimensions

DESCRIPTION Get the dimensions for this CSDT.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetFileStructure

```
virtual void GetFileStructure(ostream& os) const;
```

Output file structure BEGIN_PROLOG

NAME GetFileStructure

DESCRIPTION Output file structure description on stream. The structure of this output will be decided during later development.

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

- GetGeophysicalParameters

```
virtual RWTPtrHashSet<class RWCString>
GetGeophysicalParameters() const;
```

BEGIN_PROLOG

NAME GetGeophysicalParameters

DESCRIPTION Return array of strings describing geophysical parameters of <this> CSDT.

INPUTS none

OUTPUTS none

RETURNS RWTPtrHashSet<RWCString>

END_PROLOG

- GetInterleavingType

```
virtual RWCString GetInterleavingType() const;
```

Return interleaving used BEGIN_PROLOG

NAME GetInterleavingType

DESCRIPTION Return type of data interleaving used

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetLookUptable

```
virtual void GetLookUptable() const;
```

Return Look up Table in use BEGIN_PROLOG

NAME GetLookUptable

DESCRIPTION Return instance of GetLookUptable which describes LUT & palette used for CSDT

INPUTS none

OUTPUTS none

RETURNS void

END_PROLOG

- GetOrbitRepresentation

```
virtual RWCString GetOrbitRepresentation() const;
```

BEGIN_PROLOG

NAME GetOrbitRepresentation

DESCRIPTION The name of orbit algorithm used for this CSDT.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetRecordType

```
virtual RWCString GetRecordType() const;
```

Return record type for point data BEGIN_PROLOG

NAME GetRecordType

DESCRIPTION Return structure of CSDT record

INPUTS none

OUTPUTS none
RETURNS RWCString
END_PROLOG

- GetResolution

```
virtual long GetResolution() const;
```

BEGIN_PROLOG
NAME GetResolution
DESCRIPTION Return the resolution of this CSDT.
INPUTS none
OUTPUTS none
RETURNS long
END_PROLOG

- GetTransformationName

```
virtual RWCString GetTransformationName() const;
```

name of geometric transformation used BEGIN_PROLOG
NAME GetTransformationName
DESCRIPTION Get name of transformation used to geolocate this CSDT.
INPUTS none
OUTPUTS none
RETURNS none
END_PROLOG

- Labels

```
virtual RWTPtrHashSet<class RWCString> Labels() const;
```

Return record type for image data BEGIN_PROLOG
NAME Labels
DESCRIPTION Return labels for data object
INPUTS none
OUTPUTS none
RETURNS RWTPtrHashSet<RWCString>
END_PROLOG

- newCSDT

```
virtual GlStatus newCSDT(const RWCString& filename);
```

Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG

NAME newCSDT

DESCRIPTION Create a new CSDT of <this> type using filename provided.

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- SetLookUptable

```
virtual GlStatus SetLookUptable() const;
```

Set Look up Table to use BEGIN_PROLOG

NAME SetLookUptable

DESCRIPTION Set LookUptable which describes LUT & palette used for CSDT

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

4.3.10.3.9 DsCsImage Class

Overview:

The following classes are defined in this file: DsCsImageH The following const variables are defined in this file: none

The following defines are defined in this file: DsCsImageH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No Class definition Provides basic CSDT functionality

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsCSDT

Attributes:

Constructors and Destructor:

```
DsCsImage(const RWCString& CSDTImpType, const DsStStagingDisk&
stagingDisk, const RWCString& filename);
```

constructor

```
virtual  ~DsCsImage();
```

BEGIN_PROLOG

NAME DsCsImage

DESCRIPTION Default destructor

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

```
virtual GlStatus deleteCSDT();
```

Delete CSDT - different from destructor This will delete the file from disk aswell
BEGIN_PROLOG

NAME deleteCSDT

DESCRIPTION Destroys <this> CSDT AND deletes associated file

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractArea

```
virtual GlStatus ExtractArea(const long TLlatitude, const long
TLlongitude, const long BRlatitude, const long BRlongitude, const
long height);
```

Subset or Subsample by rectangular coords. BEGIN_PROLOG

NAME ExtractArea

DESCRIPTION Extract area of data from grid by spatial location.

INPUTS const long TLlatitude, const long TLlongitude, const long BRlatitude, const long BRlongitude, const long height

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- GetCompressionName

  `virtual RWCString GetCompressionName() const;`

  Image specific methods - File format dependent methods forwarded to implementor

  BEGIN_PROLOG

  NAME GetCompressionName

  DESCRIPTION Return name of compression algorithm used.

  INPUTS none

  OUTPUTS none

  RETURNS RWCString

  END_PROLOG

- GetCSDTImpVersionNo

  `virtual RWCString GetCSDTImpVersionNo() const;`

  Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

  Get version number of Image CSDT Implementation Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

  BEGIN_PROLOG

  NAME GetCSDTImpVersionNo

  DESCRIPTION Returns version number of point CSDT implementation.

  INPUTS none

  OUTPUTS none

  RETURNS RWCString

  END_PROLOG

- GetCSDTVersionNo

  `virtual RWCString GetCSDTVersionNo() const;`

  makes sure the derived destructors get called

  Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

  Get version number of Image CSDT Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

  BEGIN_PROLOG

  NAME GetCSDTVersionNo

  DESCRIPTION Returns version number of <this> point CSDT.

INPUTS none
OUTPUTS none
RETURNS RWCString
END_PROLOG

- GetDimensions

```
virtual RWCString GetDimensions() const;
```

Image specific methods - File format dependent methods forwarded to implementor
BEGIN_PROLOG
NAME GetDimensions
DESCRIPTION Return Dimensions of Image.
INPUTS none
OUTPUTS none
RETURNS RWCString
END_PROLOG

- GetFileStructure

```
virtual void GetFileStructure(ostream& os) const;
```

Output file structure BEGIN_PROLOG
NAME GetFileStructure
DESCRIPTION Places file structure description on ouput stream.
INPUTS none
OUTPUTS ostream
RETURNS none
END_PROLOG

- GetResolution

```
virtual long GetResolution() const;
```

BEGIN_PROLOG
NAME GetResolution
DESCRIPTION Return resolution of Image used.
INPUTS none
OUTPUTS none
RETURNS long
END_PROLOG

- ImageLabels

```
virtual RWTPtrHashSet<class RWCString> ImageLabels() const;
```

BEGIN_PROLOG
NAME ImageLabels
DESCRIPTION Return image labels.
INPUTS none
OUTPUTS none
RETURNS RWTPtrHashSet<RWCString>
END_PROLOG

- newCSDT

```
virtual GlStatus newCSDT(const RWCString& filename);
```

Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG
NAME newCSDT
DESCRIPTION Create a new CSDT with a new (empty) point file
INPUTS RWCString // filename to call new point file
OUTPUTS none
RETURNS GlStatus
END_PROLOG

4.3.10.3.10 DsCsLookupTable Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

myColorNames:

myName:

Constructors and Destructor:

Operations:

- SetColor

```
SetColor(id);
```

- SetContrast

```
SetContrast();
```

4.3.10.3.11 DsCsPoint Class

Overview:

The following classes are defined in this file: DsCsPointH

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsPointH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

Distributed Object: No Class definition Provides basic CSDT functionality

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsCSDT

Attributes:

Constructors and Destructor:

```
DsCsPoint(const RWCString& CSDTImpType, const DsStStagingDisk&
stagingDisk, const RWCString& filename);
```

constructor

```
virtual  ~DsCsPoint();
```

BEGIN_PROLOG

NAME DsCsPoint

DESCRIPTION Default destructor

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

```
virtual GlStatus deleteCSDT();
```

Delete CSDT - different from destructor This will delete the file from disk aswell BEGIN_PROLOG

NAME deleteCSDT

DESCRIPTION Destroys <this> CSDT AND deletes associated file

INPUTS none

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractSlab

```
virtual GlStatus ExtractSlab(const GlTimeP& starttime, const GlTimeP& stoptime, const int stride);
```

Subset or Subsample by time, using start time and stop time BEGIN_PROLOG

NAME ExtractSlab

DESCRIPTION Extract 'slab' of data from point by time.

INPUTS GlTimeP& - start time GlTimeP& - end time int - stride along point

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- ExtractSlab

```
virtual GlStatus ExtractSlab(const RWCString& parameterName, const int stride);
```

Subset or Subsample by parameter name BEGIN_PROLOG

NAME ExtractSlab

DESCRIPTION Extract 'slab' of data from point by parameter.

INPUTS RWCString& - name of param to extract int - stride along point

OUTPUTS none

RETURNS GlStatus

END_PROLOG

- GetCSDTImpVersionNo

```
virtual RWCString GetCSDTImpVersionNo() const;
```

Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

Get version number of Point CSDT Implementation Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetCSDTImpVersionNo

DESCRIPTION Returns version number of point CSDT implementation.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetCSDTVersionNo

```
virtual RWCString GetCSDTVersionNo() const;
```

makes sure the derived destructors get called

Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

Get version number of Point CSDT Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

BEGIN_PROLOG

NAME GetCSDTVersionNo

DESCRIPTION Returns version number of <this> point CSDT.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetDimensions

```
virtual RWCString GetDimensions() const;
```

Point specific methods - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetDimensions

DESCRIPTION Return dimensions of science data

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetFileStructure

```
virtual void GetFileStructure(ostream& os) const;
```

Output file structure BEGIN_PROLOG

NAME GetFileStructure

DESCRIPTION Places file structure description on ouput stream.

INPUTS none

OUTPUTS ostream

RETURNS none

END_PROLOG

- GetGeophysicalParameters

```
virtual RWTPtrHashSet<class RWCString>
GetGeophysicalParameters() const;
```

BEGIN_PROLOG

NAME GetGeophysicalParameters

DESCRIPTION Return searchable geophysical parameters

INPUTS

OUTPUTS none

RETURNS RWTPtrHashSet<RWCString>

END_PROLOG

- GetRecordType

```
virtual RWCString GetRecordType() const;
```

Point specific methods - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetRecordType

DESCRIPTION Return record type science data

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- newCSDT

  `virtual GlStatus newCSDT(const RWCString& filename);`

  Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG
  NAME newCSDT
  DESCRIPTION Create a new CSDT with a new (empty) point file
  INPUTS RWCString // filename to call new point file
  OUTPUTS none
  RETURNS GlStatus
  END_PROLOG

4.3.10.3.12 DsCsRaw Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:
Inherits from DsCsCSDT

Attributes:

`mylocation`:

`myname`:

`mysize`:

Constructors and Destructor:

Operations:

- GetSize

  `GetSize();`

4.3.10.3.13 DsCsSwath Class

Overview:

The following classes are defined in this file: DsCsSwathH

The following const variables are defined in this file: none

The following defines are defined in this file: DsCsSwathH

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following public functions are defined in this file:

The following private functions are defined in this file:

Distributed Object: No Class definition Provides basic CSDT functionality

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCsCSDT

Attributes:

Constructors and Destructor:

```
DsCsSwath(const RWCString& CSDTImpType, const DsStStagingDisk&
stagingDisk, const RWCString& filename);
```

constructor

```
virtual  ~DsCsSwath();
```

BEGIN_PROLOG

NAME DsCsSwath

DESCRIPTION Default destructor

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- deleteCSDT

  ```
  virtual GlStatus deleteCSDT();
  ```

  Delete CSDT - different from destructor This will delete the file from disk aswell BEGIN_PROLOG

  NAME deleteCSDT

  DESCRIPTION Destroys <this> CSDT AND deletes associated file

  INPUTS none

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

- ExtractRows

  ```
  virtual GlStatus ExtractRows(const int startRowNum, const int
  endRowNum, int actualEndRowNum, const int stride);
  ```

  Subset or Subsample by Complete Row, using start row and stop row, or start row and no. rows forward. BEGIN_PROLOG

  NAME ExtractSlab

  DESCRIPTION Extract 'slab' of data from swath by time.

  INPUTS int startRowNum - start row number int endRowNum - end row number RWBoolean actualEndRowNum - TRUE = end row no. is actual FALSE = end row no. is relative int stride - stride along swath

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

- ExtractSlab

  ```
  virtual GlStatus ExtractSlab(const GlTimeP& orbittime, const
  long duration, const int stride);
  ```

  Subset or Subsample by time, using start time and duration (in seconds) BEGIN_PROLOG

  NAME ExtractSlab

  DESCRIPTION Extract 'slab' of data from swath by time.

  INPUTS GlTimeP& - start time long - end time as seconds from start int - stride along swath

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

- ExtractSlab

  `virtual GlStatus ExtractSlab(const GlTimeP& starttime, const GlTimeP& stoptime, const int stride);`

  Subset or Subsample by time, using start time and stop time BEGIN_PROLOG

  NAME ExtractSlab

  DESCRIPTION Extract 'slab' of data from swath by time.

  INPUTS GlTimeP& - start time GlTimeP& - end time int - stride along swath

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

- ExtractSlab

  `virtual GlStatus ExtractSlab(const RWCString& parameterName, const int stride);`

  Subset or Subsample by parameter name BEGIN_PROLOG

  NAME ExtractSlab

  DESCRIPTION Extract 'slab' of data from swath by parameter.

  INPUTS RWCString& - name of param to extract int - stride along swath

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

- GetCompressionName

  `virtual RWCString GetCompressionName() const;`

  BEGIN_PROLOG

  NAME GetCompressionName

  DESCRIPTION Return name of compression algorithm used on file

  INPUTS none

  OUTPUTS none

  RETURNS RWCString

  END_PROLOG

- GetCSDTImpVersionNo

  `virtual RWCString GetCSDTImpVersionNo() const;`

  Get version number of Swath CSDT Implementation BEGIN_PROLOG

  NAME GetCSDTImpVersionNo

DESCRIPTION Returns version number of swath CSDT implementation.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetCSDTVersionNo

```
virtual RWCString GetCSDTVersionNo() const;
```

makes sure the derived destructors get called

Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

Get version number of Swath CSDT Pure Virtual methods inherited from Base Class - File format independent methods implemented by this class

BEGIN_PROLOG

NAME GetCSDTVersionNo

DESCRIPTION Returns version number of <this> swath CSDT.

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetDimensions

```
virtual RWCString GetDimensions() const;
```

BEGIN_PROLOG

NAME GetDimensions

DESCRIPTION Return array dimensions (in meters) of science data

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetFileStructure

```
virtual void GetFileStructure(ostream& os) const;
```

Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

Output file structure Pure Virtual methods inherited from Base Class - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetFileStructure

DESCRIPTION Places file structure description on output stream.

INPUTS none

OUTPUTS ostream

RETURNS none

END_PROLOG

- GetGeophysicalParameters

```
virtual RWTPtrHashSet<class RWCString>
GetGeophysicalParameters() const;
```

BEGIN_PROLOG

NAME GetGeophysicalParameters

DESCRIPTION Return searchable geophysical parameters

INPUTS none

OUTPUTS none

RETURNS RWTPtrHashSet<RWCString>

END_PROLOG

- GetOrbitRepresentation

```
virtual RWCString GetOrbitRepresentation() const;
```

Swath specific methods - File format dependent methods forwarded to implementor

Swath specific methods - File format dependent methods forwarded to implementor

BEGIN_PROLOG

NAME GetOrbitRepresentation

DESCRIPTION Return swath orbit representation name to client

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetResolution

```
virtual long GetResolution() const;
```

BEGIN_PROLOG

NAME GetResolution

DESCRIPTION Return resolution (in meters) of science data

INPUTS

OUTPUTS none

RETURNS long

END_PROLOG

- newCSDT

  `virtual GlStatus newCSDT(const RWCString& filename);`

  Create CSDT - different from constructor This will create a new file on disk BEGIN_PROLOG

  NAME newCSDT

  DESCRIPTION Create a new CSDT with a new (empty) swath file

  INPUTS RWCString // filename to call new swath file

  OUTPUTS none

  RETURNS GlStatus

  END_PROLOG

4.3.11 da Class Category

4.3.11.1 Overview

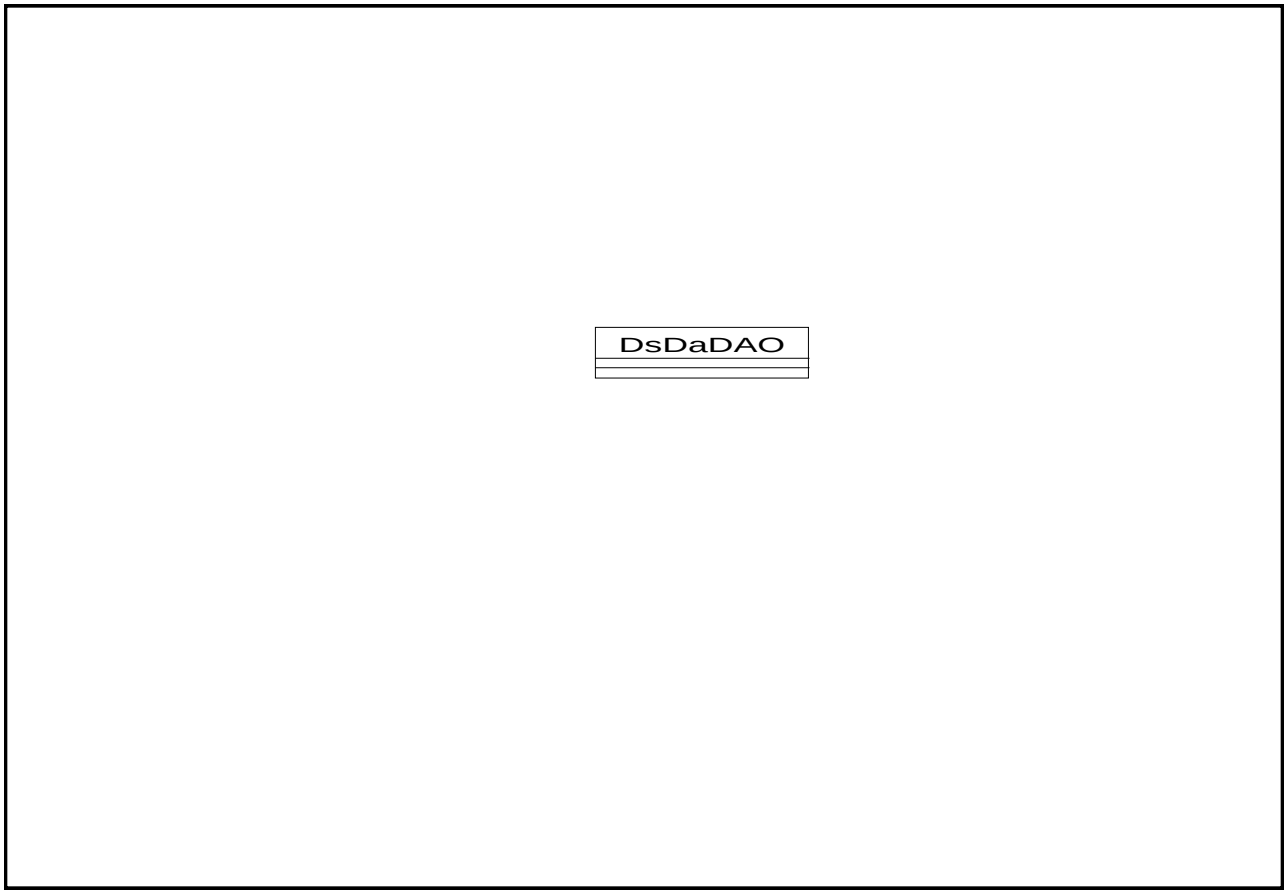


Figure 15 Main

4.3.11.2 da Classes

4.3.11.2.1 DsDaDAO Class

Overview:

Distributed Object: No This class is an abstraction of the products from the DAO.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

```
DsDaDAO();

~DsDaDAO();
```

Operations:

4.3.12 db Class Category

4.3.12.1 Overview

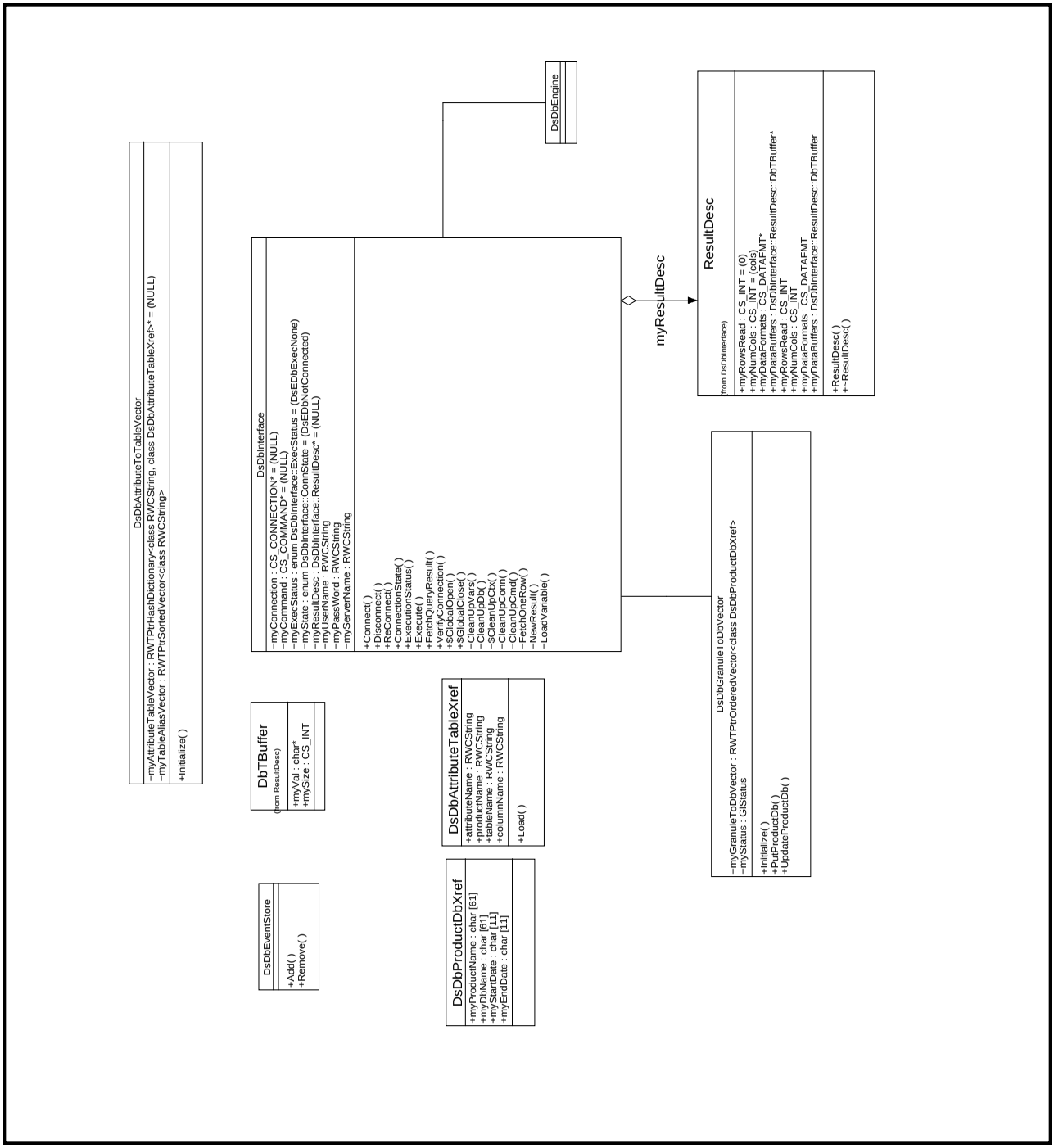


Figure 16 Main

4.3.12.2 db Classes

4.3.12.2.1 DsDbAttributeTableXref Class

Overview:

Distributed Object: No attribute table entry class

Export Control: Implementation

Inheritance Relationships:

Attributes:

`attributeName: RWCString`

`columnName: RWCString`

table name

`productName: RWCString`

attribute name

`tableName: RWCString`

product name

Constructors and Destructor:

`DsDbAttributeTableXref();`

column name constructor: set string length

Operations:

- Load

`GlStatus Load(GlParameterList& values);`

load values from a GlParameterList (result from the query) BEGIN_PROLOG

NAME Load

DESCRIPTION This method loads the state variables from the GlParameterList (result of the database query fetching).

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

GLOBALS None.

FILES None.

END_PROLOG

4.3.12.2.2 DsDbAttributeToTableVector Class

Overview:

This file contains the common methods and definitions for the DsDbAttributeTableXref and DsDbAttributeToTableVector classes.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myAttributeTableVector: RWTPtrHashDictionary<class RWCString, class DsDbAttributeTableXref>*`

hash table with attribute name as key need update when product name is used

`myTableAliasVector: RWTPtrSortedVector<class RWCString>`

alias table

Constructors and Destructor:

`DsDbAttributeToTableVector();`

destructor

`~DsDbAttributeToTableVector();`

destructor BEGIN_PROLOG

NAME ~DsDbAttributeToTableVector

DESCRIPTION This destructor method deallocates all the in-memory hash table entries

RETURNS None.

GLOBALS None.

FILES None.

END_PROLOG

Operations:

- GetAttrTableXref

  `GlStatus GetAttrTableXref(const RWCString& attrName, DsDbAttributeTableXref*& attrTableXref) const;`

  get attribute table entry given attribute name BEGIN_PROLOG

  NAME GetAttrTableXref

  DESCRIPTION This method finds the hash table entry given the attribute name. NULL will be returned if nothing is found.

  RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  GLOBALS None.

  FILES None.

  NOTES Currently, the assumption is that the attribute name uniquely identifies a column and table. If this is relaxed, then more parameters are needed to retrieve the table entry.

  END_PROLOG

- GetTableAlias

  `GlStatus GetTableAlias(const RWCString& tableName, RWCString& aliasName) const;`

  get alias given table name BEGIN_PROLOG

  NAME GetTableAlias

  DESCRIPTION This method finds a table name's alias given the table name. The alias format is a prefix followed by the table name's index number. (e.g. a12; a is the prefix and 12 is the index number)

  RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message. GLOBALS None.

  FILES None.

  END_PROLOG

- Initialize

  `GlStatus Initialize(DsDbInterface* connection);`

  initialize the table (load everything from database BEGIN_PROLOG

  NAME Initialize

  DESCRIPTION This method initilizes the in-memory hash table and the attribute alias table. It first sets the size of the hash table and loads it with all the attributes. It then loads the alias table for each of the table names.

  RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

GLOBALS None.
FILES None.
END_PROLOG

4.3.12.2.3 DsDbEngine Class

Overview:
This class represents the Sybase SQL server COTS product.
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.12.2.4 DsDbEventStore Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

- Add

```
Add();
```

- GetEvents

```
GetEvents();
```

- GetEvent

```
GetEvent();
```

- GetNextID

```
GetNextID();
```

- Remove

```
Remove();
```

4.3.12.2.5 DsDbGranuleToDbVector Class

Overview:

This file contains the common functions and definitions for the DsDbProductDbXref and the DsDbGranuleToDbVector classes.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myGranuleToDbVector: RWTPtrOrderedVector<class
DsDbProductDbXref>
```

```
myStatus: GlStatus
```

Constructors and Destructor:

```
DsDbGranuleToDbVector();
```

```
~DsDbGranuleToDbVector();
```

Operations:

- GetProductDb

```
GlStatus GetProductDb(DsDbProductDbXref& ,
RWTPtrOrderedVector<class DsDbProductDbXref>& , DsDbInterface&
);
```

- Initialize

```
GlStatus Initialize(DsDbInterface* );
```

- PutProductDb

```
GlStatus PutProductDb();
```

- UpdateProductDb

```
GlStatus UpdateProductDb();
```

4.3.12.2.6 DsDbInterface Class

Overview:

This file contains the common methods and definitions for the DsDbInterface class.

sybase client library version number Distributed Object: No

db interface class

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myCommand: CS_COMMAND*`

sybase open client connection

`myConnection: CS_CONNECTION*`

database server name sybase specific data structures CS_CONTEXT* myContext; // sybase open client db context

`myExecStatus: enum DsDbInterface::ExecStatus`

sybase open client command

`myPassWord: RWCString`

user name for database connection

`myResultDesc: DsDbInterface::ResultDesc*`

`myServerName: RWCString`

password for database conenction

`myState: enum DsDbInterface::ConnState`

current query exeuction status

`myUserName: RWCString`

Constructors and Destructor:

`DsDbInterface(void );`

constructor

`~DsDbInterface(void );`

destructor BEGIN_PROLOG

NAME ~DsDbInterface

DESCRIPTION

This destructor function calls functions necessary to clean up and deallocate all the data structures.

RETURNS void.

GLOBALS None.

FILES None.

END_PROLOG

Operations:

- CleanUpCmd

  ```
  void CleanUpCmd();
  ```

  BEGIN_PROLOG

  NAME CleanUpCmd

  DESCRIPTION

  This function deallocates and cleans up the command structure.

  RETURNS void.

  GLOBALS None.

  FILES None.

  END_PROLOG

- CleanUpConn

  ```
  void CleanUpConn();
  ```

  BEGIN_PROLOG

  NAME CleanUpConn

  DESCRIPTION

  This function closes and cleans up the database connection.

  RETURNS void.

  GLOBALS None.

  FILES None.

  END_PROLOG

- CleanUpCtx

  ```
  static void CleanUpCtx();
  ```

  BEGIN_PROLOG

  NAME CleanUpCtx

  DESCRIPTION

This function deallocates and cleans up the database context structure.

RETURNS void.

GLOBALS None.

FILES None.

END_PROLOG

- CleanUpDb

```
void CleanUpDb();
```

BEGIN_PROLOG

NAME CleanUpDb

DESCRIPTION

This function calls functions necessary to clean up database related structures: command, connection and context.

RETURNS void.

GLOBALS None.

FILES None.

END_PROLOG

- CleanUpVars

```
void CleanUpVars();
```

result descriptor set of cleanup routines BEGIN_PROLOG

NAME CleanUpVars

DESCRIPTION

This function removes (nullifies) all connection parameters.

RETURNS void.

GLOBALS None.

FILES None.

END_PROLOG

- ConnectionState

```
enum DsDbInterface::ConnState ConnectionState(void );
```

report current connection status BEGIN_PROLOG

NAME ConnectionState

DESCRIPTION

This function provides the connection state.

RETURNS ConnState Enumerated connection status.

GLOBALS None.

FILES None.

END_PROLOG

- Connect

```
GlStatus Connect(const RWCString& userName, const RWCString&
password, const RWCString& serverName);
```

establish database connection BEGIN_PROLOG

NAME Connect

DESCRIPTION This function establishes database connection by first dropping any existing connections, then setting up the interface and allocating a connection structure, setting up diagnose options and connection properties. State variables are also reset to reflect true state of connection.

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

GLOBALS SYBINTERFACES environment variable is read and must exist in user environment.

FILES Errors written to cout.

END_PROLOG

- Disconnect

```
GlStatus Disconnect(void );
```

close database connection BEGIN_PROLOG

NAME Disconnect

DESCRIPTION

This function closes and cleans up the database connection if one already exists.

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

GLOBALS None.

FILES Errors written to cout.

END_PROLOG

- Execute

```
enum DsDbInterface::ExecStatus Execute(const RWCString& SQLCmd);
```

execute SQL query BEGIN_PROLOG

NAME Execute

DESCRIPTION

This function executes an SQL statement if a connection to the database exists.

RETURNS ExecStatus Enumerated SQL query execution status.

GLOBALS None.

FILES Errors written to cout.

END_PROLOG

- ExecutionStatus

```
enum DsDbInterface::ExecStatus ExecutionStatus(void );
```

report current query execution status BEGIN_PROLOG

NAME ExecutionStatus

DESCRIPTION

This function provides the execution status of an SQL statement.

RETURNS ExecStatus Enumerated SQL query execution status.

GLOBALS None.

FILES None.

END_PROLOG

- FetchOneRow

```
enum DsDbInterface::ExecStatus FetchOneRow();
```

for fetching results BEGIN_PROLOG

NAME FetchOneRow

DESCRIPTION This function fetches a single row from the query result into the binding data buffer (result descriptor).

RETURNS ExecStatus Enumerated SQL fetch execution status.

GLOBALS None.

FILES None.

NOTE FetchOneRow is recursive. Recursion is used to fetch multiple result sets from the database. Could also be implemented with use of double nested loops.

END_PROLOG

- FetchQueryResult

```
enum DsDbInterface::ExecStatus
FetchQueryResult(GlParameterList& result);
```

fetch query result given a GlParameterList BEGIN_PROLOG

NAME FetchQueryResult

DESCRIPTION This overloaded method calls FetchOneRow to populate the binding area and then calls LoadVariable to populate the Gl Parameter from the binding area.

RETURNS ExecStatus Enumerated SQL fetch execution status.

GLOBALS None.

FILES None.

END_PROLOG

- FetchQueryResult

```
enum DsDbInterface::ExecStatus
FetchQueryResult(RWTPtrOrderedVector<void*>& result);
```

fetch query result given pre-allocated space BEGIN_PROLOG

NAME FetchQueryResult

DESCRIPTION This overloaded method calls FetchOneRow to populate the binding area and then calls LoadVariable to populate the result vector array from the binding area.

RETURNS ExecStatus Enumerated SQL fetch execution status.

GLOBALS None.

FILES None.

END_PROLOG

- GlobalClose

```
static void GlobalClose();
```

BEGIN_PROLOG

NAME DsDbInterface::GlobalOpen

DESCRIPTION This method is used to release the static Sybase context structure to be used by all connections.

RETURNS void

GLOBALS the static Sybase context myContext.

FILES None.

END_PROLOG

- GlobalOpen

```
static GlStatus GlobalOpen();
```

BEGIN_PROLOG

NAME DsDbInterface::GlobalOpen

DESCRIPTION This method is used to setup the static Sybase context structure to be used by all connections.

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

GLOBALS the static Sybase context myContext.

FILES None.

END_PROLOG

- LoadVariable

```
GlStatus LoadVariable(GlParameter* var, int offset);
```

for variable bindings BEGIN_PROLOG

NAME LoadVariable

DESCRIPTION

This overloaded method binds values from the result description data buffer to the appropriate Gl Paramter (based on data type).

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message. GLOBALS None.

FILES Errors written to cout.

END_PROLOG

- LoadVariable

```
GlStatus LoadVariable(void* var, int offset);
```

BEGIN_PROLOG

NAME LoadVariable

DESCRIPTION This overloaded method binds values from the result description data buffer to pre-allocated structures.

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message. GLOBALS None.

FILES Errors written to cout.

NOTE var is bound at runtime.

END_PROLOG

- NewResult

```
enum DsDbInterface::ExecStatus NewResult(int ignore);
```

BEGIN_PROLOG

NAME NewResult

DESCRIPTION

This function sets up the result descriptors only if SQL statement has successfully executed.

RETURNS ExecStatus Enumerated SQL query execution status.

GLOBALS None.

FILES Errors are written to cout.

END_PROLOG

- ReConnect

```
GlStatus ReConnect(void );
```

re-establish database connection BEGIN_PROLOG

NAME ReConnect

DESCRIPTION

This function re-establishes the database connection.

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message. GLOBALS None.

FILES Errors written to cout.

END_PROLOG

- VerifyConnection

```
GlStatus VerifyConnection(void );
```

verify current database connection BEGIN_PROLOG

NAME VerifyConnection

DESCRIPTION This function is used to verify the database connection. If a connection does not exist, then re-connect.

RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

GLOBALS None.

FILES None.

END_PROLOG

4.3.12.2.7 DsDbProductDbXref Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myDbName: char [61]
```

```
myEndDate: char [11]
```

```
myProductName: char [61]

myStartDate: char [11]
```

Constructors and Destructor:

```
DsDbProductDbXref();

 ~DsDbProductDbXref();
```

Operations:

- operator ==

```
int operator ==(const DsDbProductDbXref& );
```

4.3.13 de Class Category

4.3.13.1 Overview

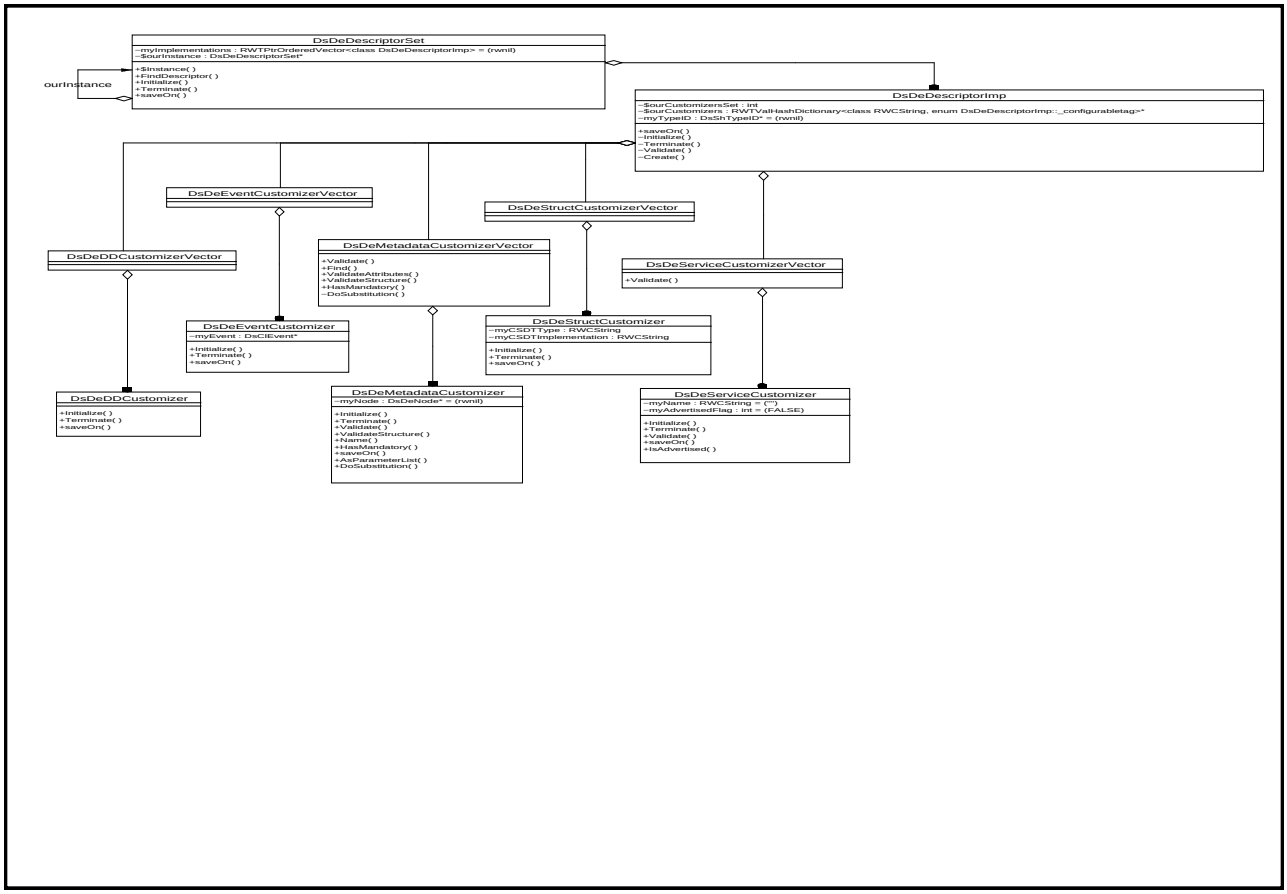


Figure 17 DsDe1

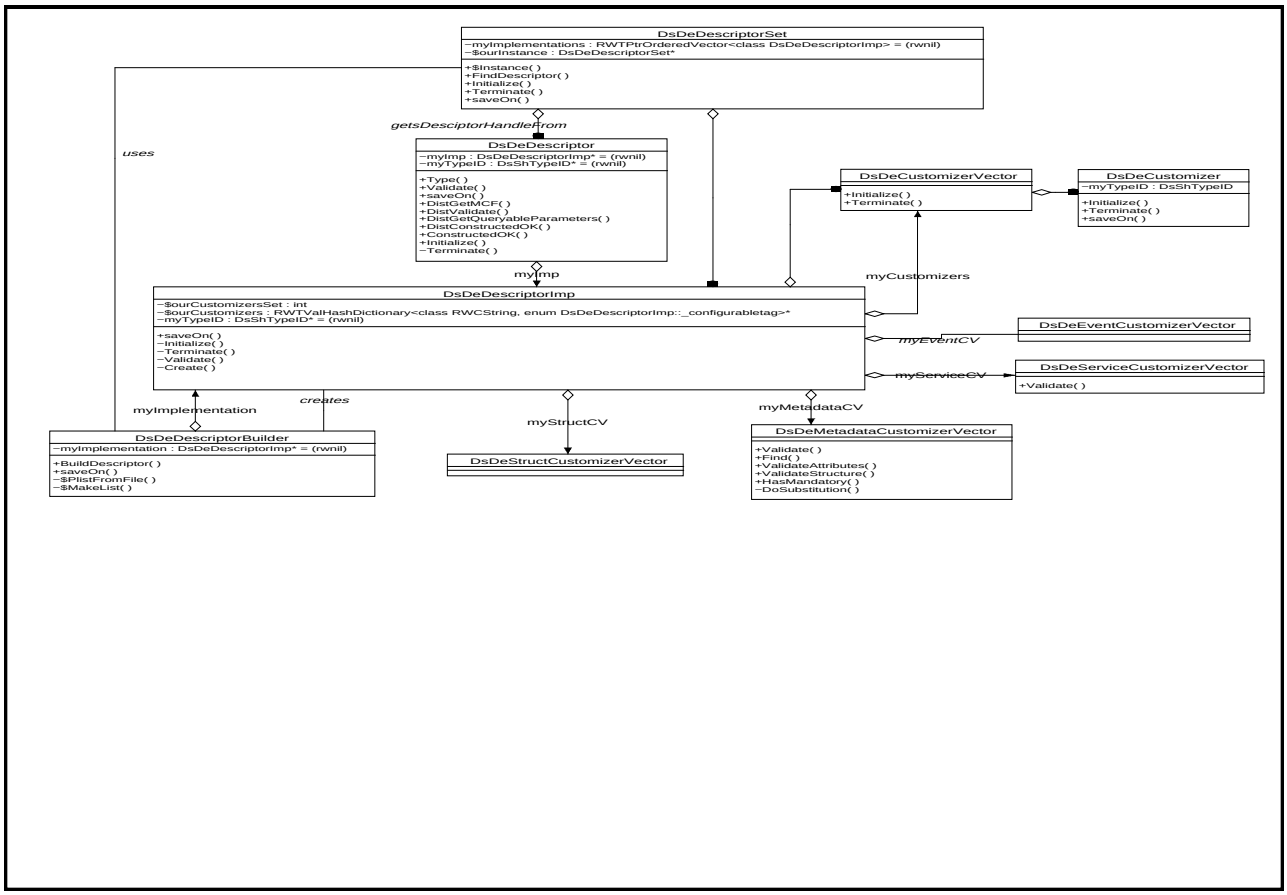


Figure 18 DsDe2

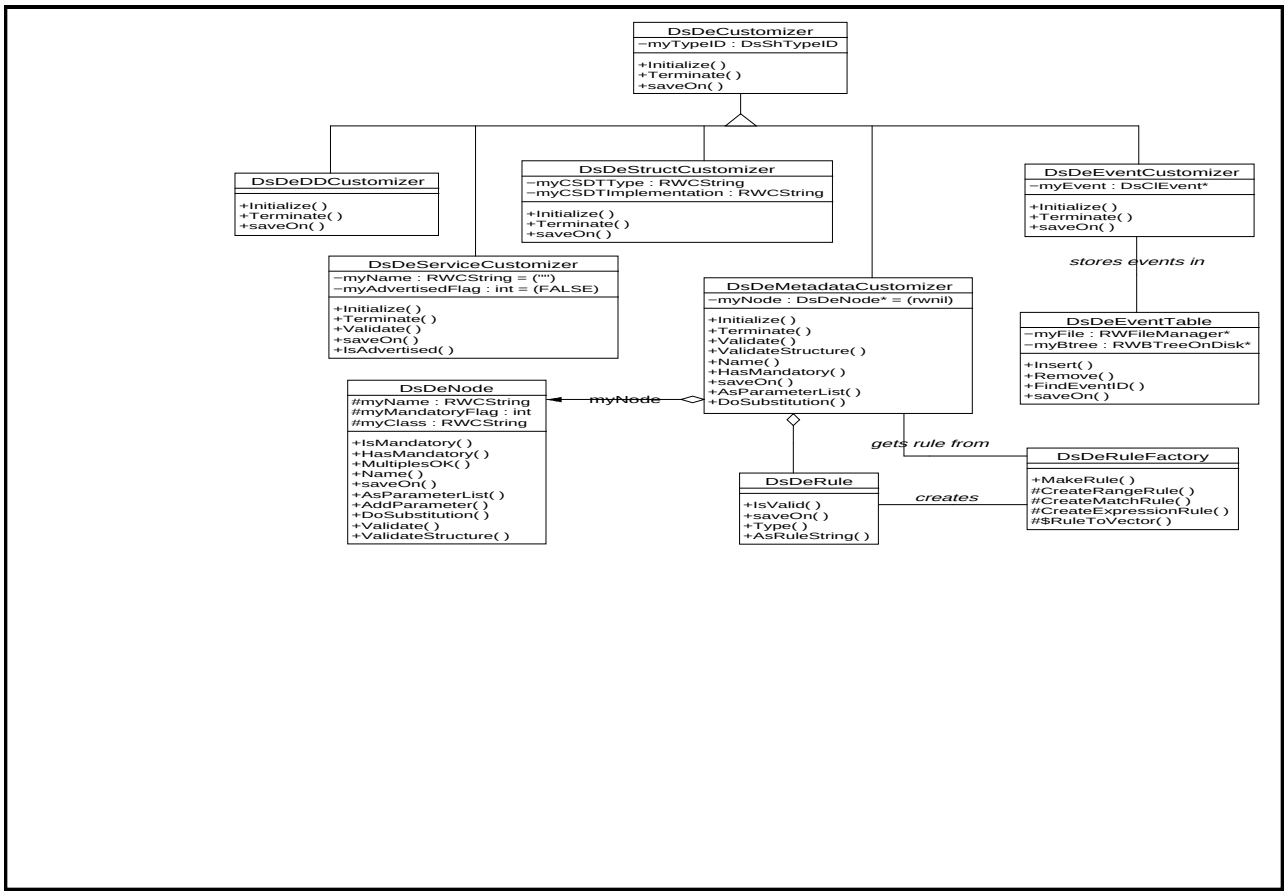


Figure 19 DsDe3

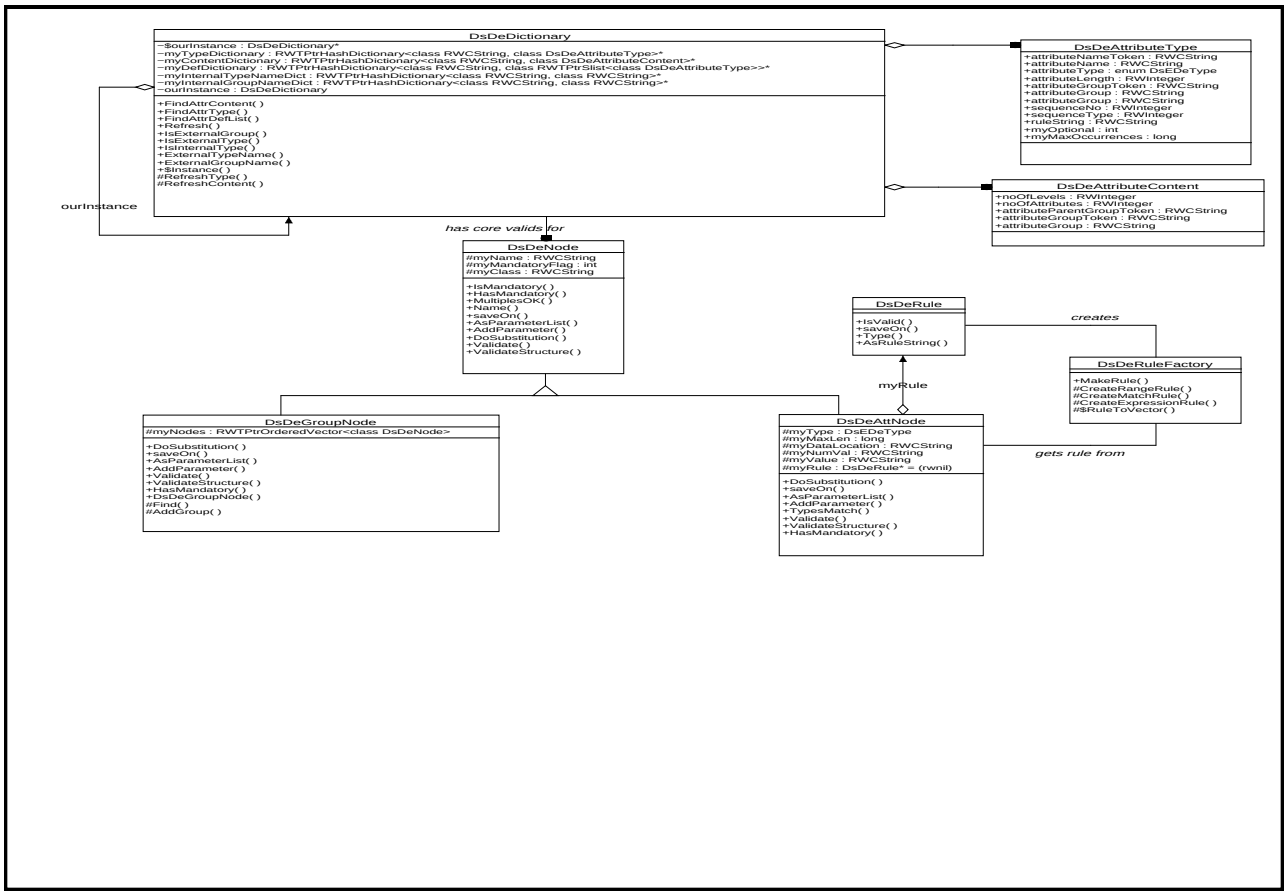


Figure 20 DsDe4

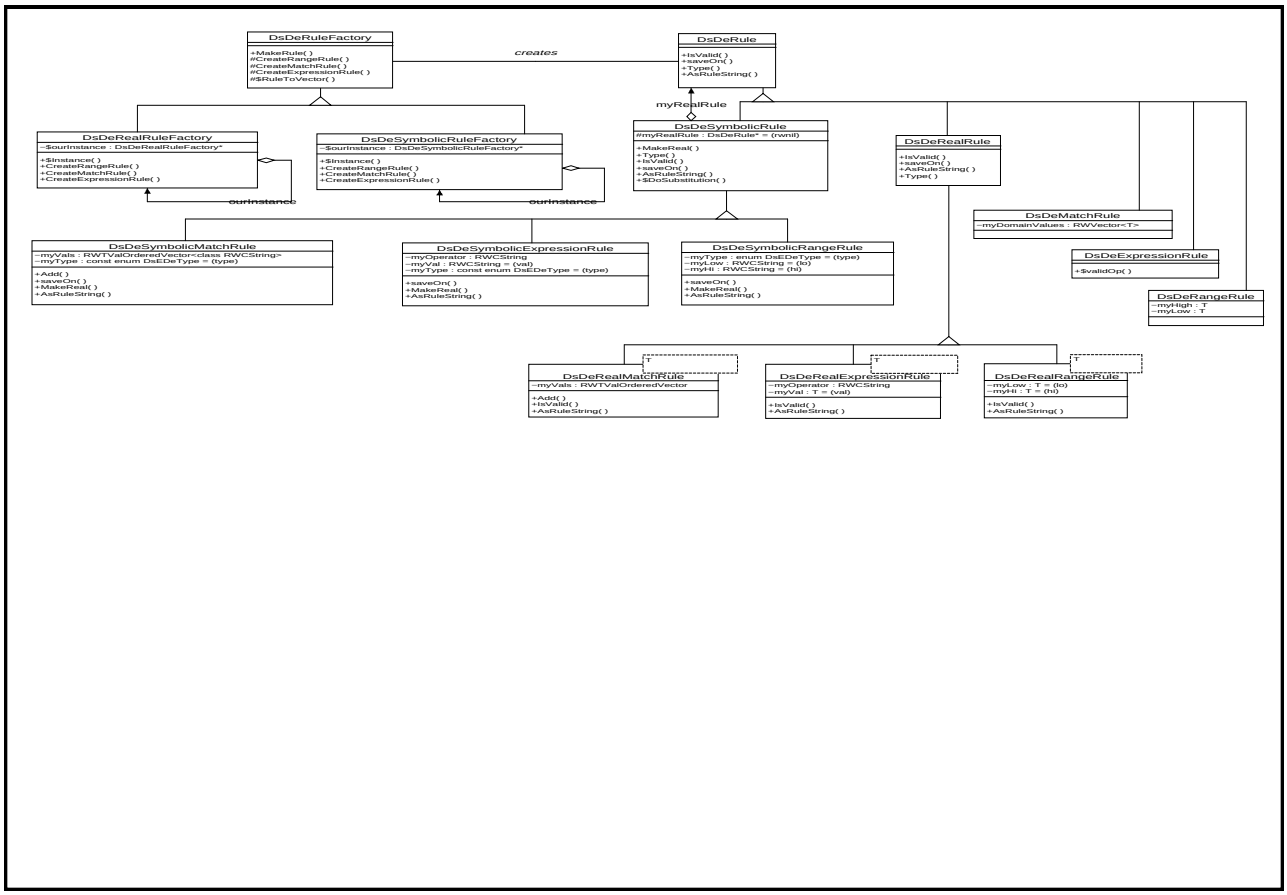


Figure 21 DsDe5

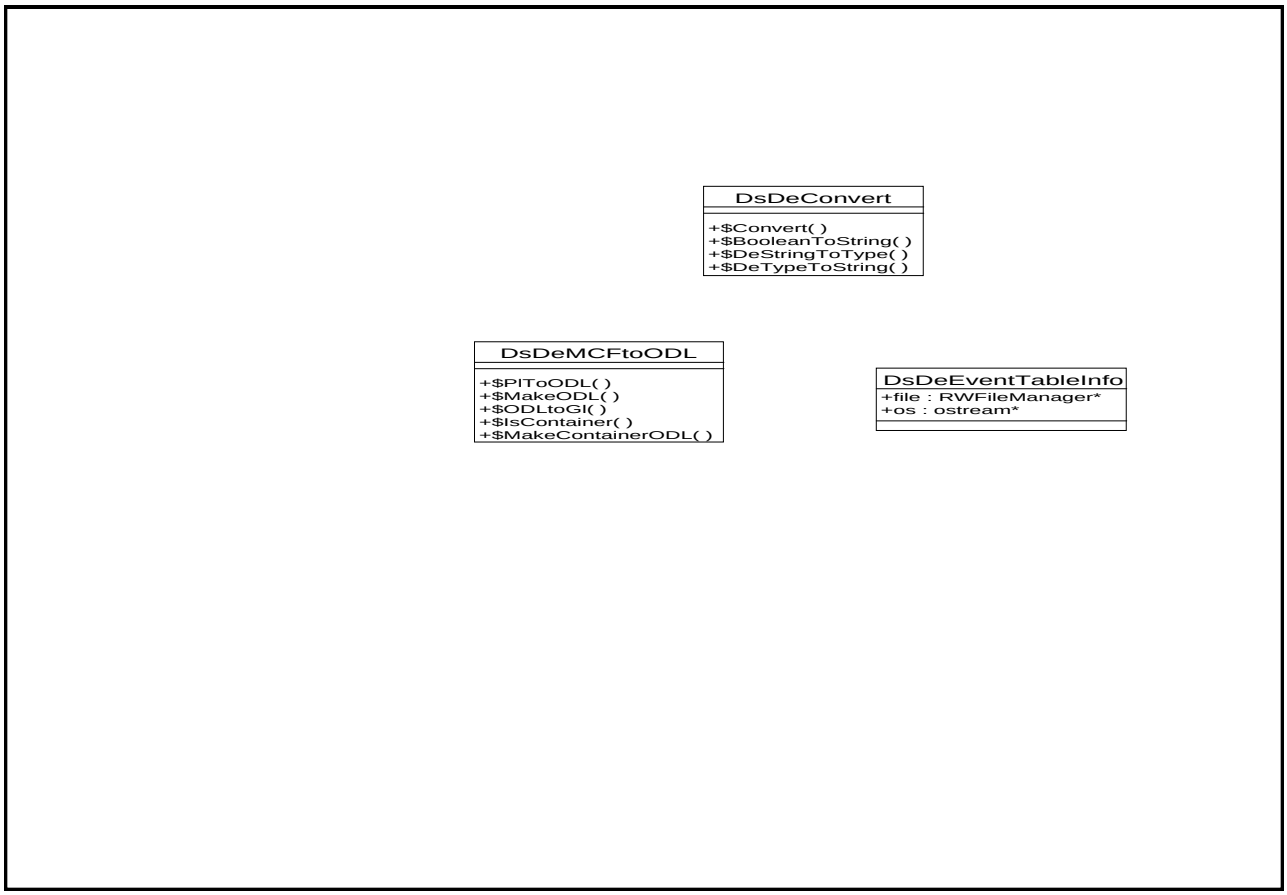


Figure 22 DsDe6

4.3.13.2 de Classes

4.3.13.3 DsDeAttNode Class

Overview:

This class is used to hold a grouped set of attributes

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeNode

Attributes:

`myDataLocation: RWCString`

maximum length of attribute

`myMaxLen: long`

type, i.e. int, long, etc.

`myNumVal: RWCString`

where value obtained from

`myRule: DsDeRule*`

value if this is collection attribute

`myType: DsEDeType`

`myValue: RWCString`

number of values in sequence

Constructors and Destructor:

`DsDeAttNode(const DsDeAttNode& );`

create from dd

`DsDeAttNode(const RWCString& name);`

ParameterList containing

`DsDeAttNode(const RWCString& name, const GlParameterList& pl);`

`~DsDeAttNode();`

Output. results/errors BEGIN_PROLOG

NAME ~DsDeAttNode

DESCRIPTION This destructor deletes the memory allocated by this class.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AddParameter

`void AddParameter(GlParameterList& pl);`

make parameterlist of atts BEGIN_PROLOG

NAME AddParameter

DESCRIPTION This member function is used to add an empty parameter of the same type and name as this attribute to the given parameterlist.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- AsParameterList

```
GlParameterList* AsParameterList();
```

BEGIN_PROLOG

NAME AsParameterList

DESCRIPTION This member function creates its contents in the form of a parameter list.

RETURNS N/A

GLOBALS None

FILES None

NOTES The user of this class is responsible for recursively deleting this list. END_PROLOG

- DoSubstitution

```
void DoSubstitution(const GlParameterList& pl);
```

assignment operator BEGIN_PROLOG

NAME DoSubstitution

DESCRIPTION This member function checks to see if a rule exists for this attribute. If the rule exists and it is a symbolic rule, it is turned into a real rule.

RETURNS void

GLOBALS None

FILES None

END_PROLOG

- HasMandatory

```
int HasMandatory(const GlParameterList& pl, GlStatus& stat)
const;
```

Results BEGIN_PROLOG

NAME HasMandatory

DESCRIPTION This member function determines whether this attribute is mandatory. If it is not, then return true. Otherwise, check to see if this attribute is in the given parameterlist. If it is not, add an error message to the given status object.

RETURNS RWBoolean indicating whether the parameterlist has mandatory attribute

GLOBALS None

FILES None

END_PROLOG

- operator =

```
DsDeAttNode& operator =(const DsDeAttNode& );
```

copy constructor

- saveOn

```
void saveOn(ostream& os) const;
```

input. real values for substitution BEGIN_PROLOG

NAME saveOn

DESCRIPTION This member function is used for debugging purposes. It prints out the contents of this instance.

RETURNS Nothing

GLOBALS None

FILES None

END_PROLOG

- TypesMatch

```
int TypesMatch(const GlParameter& p) const;
```

add parm of right type to plist BEGIN_PROLOG

NAME TypesMatch

DESCRIPTION This member function is used to determine whether the given parameter type matches this attributes type.

RETURNS RWBoolean indicating whether the types match or not.

GLOBALS None

FILES None

END_PROLOG

- ValidateStructure

```
virtual void ValidateStructure(const GlParameter& inP,
GlParameterList& outP, GlStatus& stat);
```

Results BEGIN_PROLOG

NAME ValidateStructure

DESCRIPTION This member function is used to validate the structure of the given parameter with respect to the output parameterlist. If multiples of this attribute are allowed, just add it to

the output list. If not, check to see if this attribute is already on the list. If not, add it. Otherwise add an error message to the GlStatus object.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- Validate

```
virtual void Validate(const GlParameter& inP, GlParameterList&
outp, GlStatus& stat);
```

BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is used to validate the given parameter and if it is valid, add it to the output parameterlist. If the parameter is not valid, an error message is added to the given status object.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.1 DsDeAttributeContent Class

Overview:

This file contains the definitions the DsMdMSAttributeContent class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`attributeGroupToken: RWCString`

`attributeGroup: RWCString`

external attribute group name

`attributeParentGroupToken: RWCString`

no. of attributes

`noOfAttributes: RWInteger`

no. of levels

`noOfLevels: RWInteger`

internal attribute group name

Constructors and Destructor:

Operations:

4.3.13.3.2 DsDeAttributeType Class

Overview:

This file contains the definitions the DsMdMSAttributeType class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`attributeGroupToken: RWCString`

attribute length

`attributeGroup: RWCString`

external group name

`attributeLength: RWInteger`

attribute type

`attributeNameToken: RWCString`

`attributeName: RWCString`

external attribute name

attributeType: enum DsEDeType

internal attribute name

myMaxOccurrences: long

myOptional: int

ruleString: RWCString

sequence type

sequenceNo: RWInteger

internal attribute group

sequenceType: RWInteger

sequence No

Constructors and Destructor:

Operations:

4.3.13.3.3 DsDeConvert Class

Overview:

This class encapsulates static functions that are used for doing various types of conversions that are needed by the descriptor. All of the functions are static and the constructor is hidden to prevent anyone from instantiating this class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeConvert();
```

hidden constructor

Operations:

- BooleanToString

```
static RWCString BooleanToString(const int& flag);
```

input. string "TRUE" or "FALSE" BEGIN_PROLOG

NAME BooleanToString

DESCRIPTION This function returns a string based on the given flag.

RETURNS RWCString

GLOBALS None

FILES None

END_PROLOG

- Convert

```
static int Convert(const RWCString& inS, double* d);
```

output. string converted to int BEGIN_PROLOG

NAME Convert

DESCRIPTION This function is used to convert the given string to a double. If there is a problem doing this, the return will be FALSE.

RETURNS Flag indicating success of the conversion or not

GLOBALS None

FILES None

END_PROLOG

- Convert

```
static int Convert(const RWCString& inS, long* i);
```

input. type i.e., int, long BEGIN_PROLOG

NAME Convert

DESCRIPTION This function is used to convert the given string to an integer. If there is a problem doing this, the return will be FALSE.

RETURNS Flag indicating success of the conversion or not

GLOBALS None

FILES None

END_PROLOG

- DeStringToType

```
static enum DsEDeType DeStringToType(const RWCString& str);
```

input. flag BEGIN_PROLOG

NAME DeStringToType

DESCRIPTION This function is used to create the enumerated type given a string.

RETURNS enumerated valud

GLOBALS None

FILES None

END_PROLOG

- DeTypeToString

```
static RWCString DeTypeToString(enum DsEDeType theType);
```

BEGIN_PROLOG

NAME DeTypeToString

DESCRIPTION This function converts the given enumerated type to a string.

RETURNS RWCString

GLOBALS None

FILES None

END_PROLOG

- SetFlag

```
static int SetFlag(int& flag, const RWCString& val);
```

output. string converted to double BEGIN_PROLOG

NAME SetFlag

DESCRIPTION This function uses the given string as the value to set the boolean flag to. If the input string is not one of the valid ones, then failure is returned.

RETURNS MCF-ified RWCString

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.4 DsDeCustomizerVector Class

Overview:

This is a container class for customizers for a given datatype. A metadata customizer provides the ability to configure the attributes that a given datatype has. It also provides validation, queryable parameters, and MCF services.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DsDeCustomizerVector();`

BEGIN_PROLOG

NAME DsDeCustomizerVector

DESCRIPTION Constructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`DsDeCustomizerVector(const DsDeCustomizerVector& );`

copy constructor - NOT IMPLEMENTED

`~DsDeCustomizerVector();`

terminate my contents BEGIN_PROLOG

NAME ~DsDeCustomizerVector

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

NOTES The vector of customizers is not owned by this class

END_PROLOG

Operations:

- Initialize

  `void Initialize();`

  default constructor BEGIN_PROLOG

  NAME Initialize

  DESCRIPTION This method iterates over the set of customizers performing initialization.

  RETURNS void

  GLOBALS None

  FILES None

  END_PROLOG

- operator =

  `DsDeCustomizerVector& operator =(const DsDeCustomizerVector& );`

  assignment operator - NOT IMPLEMENTED

- Terminate

  `void Terminate();`

  initialize my contents BEGIN_PROLOG

  NAME Terminate

  DESCRIPTION This method iterates over the set of customizers performing termination.

  RETURNS void

  GLOBALS None

  FILES None

  END_PROLOG

4.3.13.3.5 DsDeCustomizer Class

Overview:

Abstract base class that provides the ability to customize features of an event.

This is a container class for customizers for a given datatype. A metadata customizer provides the ability to configure the attributes that a given datatype has. It also provides validation, queryable parameters, and MCF services.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myTypeID: DsShTypeID
```

Constructors and Destructor:

```
DsDeCustomizer(const DsDeCustomizer& );
```

Construct with given ESDT type

```
DsDeCustomizer(const DsShTypeID& TypeID);
```

Default constructor BEGIN_PROLOG

NAME DsDeCustomizer

DESCRIPTION The constructor for this base class that takes on an ESDT TypeID

RETURNS none

END_PROLOG

```
DsDeCustomizer();
```

BEGIN_PROLOG

NAME DsDeCustomizer

DESCRIPTION The default constructor for this base class

RETURNS none

END_PROLOG

```
virtual  ~DsDeCustomizer();
```

Unimplemented BEGIN_PROLOG

NAME ~DsDeCustomizer

DESCRIPTION The destructor for this base class

RETURNS none

END_PROLOG

Operations:

- GetType

```
DsShTypeID GetType() const;
```

prints readable output to an ostream. BEGIN_PROLOG

NAME GetType

DESCRIPTION Returns the DsShTypeID of this base class

RETURNS DsShTypeID

END_PROLOG

- Initialize

```
virtual EcUtStatus Initialize() = 0;
```

Default destructor

- operator ==

```
int operator ==(const DsDeCustomizer& );
```

Pure virtual termination of event BEGIN_PROLOG

NAME operator==

DESCRIPTION Comparison operator needed so that this class can be used with rogue wave vectors.

RETURNS Always returns FALSE

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& ) const = 0;
```

Right-hand side

- SetType

```
void SetType(const DsShTypeID& TypeID);
```

BEGIN_PROLOG

NAME SetType

DESCRIPTION Sets a DsShTypeID

RETURNS none

END_PROLOG

- Terminate

```
virtual EcUtStatus Terminate() = 0;
```

Pure virtual initilization of event

4.3.13.3.6 DsDeDDCustomizerVector Class

Overview:

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeDDCustomizerVector(istream & from);
```

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.

```
~DsDeDDCustomizerVector();
```

The destructor destroys all of the objects in its contents and then destroys itself.

Operations:

4.3.13.3.7 DsDeDDCustomizer Class

Overview:

Provides the ability to customize the data dictionary information for a datatype.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeCustomizer

Attributes:

Constructors and Destructor:

```
DsDeDDCustomizer(const GlParameterList& , const DsShTypeID&
TypeID);
```

```
DsDeDDCustomizer(const DsDeDDCustomizer& );
```

copy constructor - NOT IMPLEMENTED

```
~DsDeDDCustomizer();
```

The ESDT type. BEGIN_PROLOG

NAME ~DsDeDDCustomizer

DESCRIPTION The destructor for this class

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- Initialize

```
EcUtStatus Initialize();
```

Destructor BEGIN_PROLOG

NAME Initialize

DESCRIPTION This function does nothing. It is required because it is declared as pure virtual in the base class

INPUTS none

OUTPUTS none

RETURNS EcUtStatus of the operation (OK)

END_PROLOG

- operator =

```
DsDeDDCustomizer& operator =(const DsDeDDCustomizer& );
```

assignment operator - NOT IMPLEMENTED

- saveOn

```
void saveOn(ostream& os) const;
```

Does nothing. Required by base class. BEGIN_PROLOG

NAME saveOn

DESCRIPTION Out puts the contents of this class in a readable format

INPUTS ostream& stream that out put is to be sent to

OUTPUTS stream specified by ostream& argument

RETURNS none

END_PROLOG

- Terminate

  `EcUtStatus Terminate();`

  Exports DD information BEGIN_PROLOG

  NAME Terminate

  DESCRIPTION This function does nothing. it is required because it is declared as pure virtual in the base class.

  INPUTS none

  OUTPUTS none

  RETURNS EcUtStatus of the operation (OK)

  END_PROLOG

4.3.13.3.8 DsDeDescriptorBuilder Class

Overview:

This class reads in the ASCII file that defines the descriptor for a particular type. This is the class that must be concerned with the format of this file. It isolates the parsing of ODL to this class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myImplementation: DsDeDescriptorImp*`

member variables

Constructors and Destructor:

`DsDeDescriptorBuilder();`

`DsDeDescriptorBuilder(const DsDeDescriptorBuilder& );`

Input: node to parse the following are defined by NOT IMPLEMENTED to ensure they aren't used

Operations:

- BuildDescriptor

  ```
  DsDeDescriptorImp* BuildDescriptor(const DsShTypeID& theType);
  ```

  default constructor BEGIN_PROLOG

  NAME BuildDescriptor

  DESCRIPTION This member function reads the ASCII file associated with the given TypeID and creates the right kind of customizers

  INPUTS theType - identifies the type to build

  OUTPUTS None

  RETURNS a pointer to the DsDeDescriptorImp object that contains the customizers

  GLOBALS The name of the environment variable that has the name of the configuration file

  FILES A configuration file that contains the name of the descriptor files for each configured type

  END_PROLOG

- MakeList

  ```
  static Aggregate_Node* MakeList(Aggregate_Node* startAt,
  GlParameterList* to, Aggregate_Node* from);
  ```

  Output: results BEGIN_PROLOG

  NAME MakeList

  DESCRIPTION This static function parses a branch in an ODL tree and produces a parameter list. Parameters within and OBJECT become GlStringPs and GROUPS become parameter lists. This function is called recursively to handle GROUPS within GROUPS.

  RETURNS GlStatus indication of success or failure

  FILES A descriptor file that contains the ODL formatted data for a type

  END_PROLOG

- operator =

  ```
  DsDeDescriptorBuilder& operator =(const DsDeDescriptorBuilder&
  );
  ```

  copy constructor - NOT IMPLEMENTED

- PlistFromFile

  ```
  static GlStatus PlistFromFile(const RWCString& theFile,
  GlParameterList* outList);
  ```

  BEGIN_PROLOG

  NAME PlistFromFile

DESCRIPTION This static function reads the given ASCII file which is in ODL format and creates a parameterList

INPUTS theFile - the ODL formatted file

OUTPUTS outList - the resulting parameterList

RETURNS GlStatus indication of success or failure

GLOBALS

FILES A descriptor file that contains the ODL formatted data for a type

END_PROLOG

- saveOn

  `void saveOn(ostream& os) const;`

  Input: type of the descriptor to create BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION This member function is used for debug purposes. It allows you to print out the contents of this instance while in the debugger.

  RETURNS N/A

  GLOBALS None

  FILES None

  END_PROLOG

4.3.13.3.9 DsDeDescriptorImp Class

Overview:

This class contains all of the attributes and associated implementation of the descriptor class. The interface is separated from the implementation so that many interfaces can use a single implementation. This is done because descriptor implementations are expensive to create due to the parsing of information.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myCustomizers: DsDeCustomizerVector`

data type of this descriptor

myEventCV: RWTPtrOrderedVector<class DsDeEventCustomizer>

vector of all customizers

myMetadataCV: DsDeMetadataCustomizerVector

vector of event customizers

myServiceCV: DsDeServiceCustomizerVector

vector of metadata customizers

myStructCV: DsDeStructCustomizerVector

vector of service customizers

myTypeID: DsShTypeID*

maps customizer names (strings) to enum

ourCustomizersSet: int

initializes static customizer table

ourCustomizers: RWTValHashDictionary<class RWCString, enum DsDeDescriptorImp::_configurabletag>*

flags whether table initialized yet

Constructors and Destructor:

DsDeDescriptorImp(const DsShTypeID& type, const GlParameterList& theGroups);

DsDeDescriptorImp(const DsDeDescriptorImp&);

vector of structure customizers the following are declared by NOT IMPLEMENTED to prevent their use

~DsDeDescriptorImp();

Output: Write output to this stream BEGIN_PROLOG

NAME ~DsDeDescriptorImp

DESCRIPTION Destructor for this class. Deallocates previously new'd memory.

RETURNS none

END_PROLOG

Operations:

- Create

```
void Create(const GlParameterList& inList, const DsShTypeID&
theType);
```

Get whether EOSHDF, GRIB, etc. BEGIN_PROLOG

NAME Create

DESCRIPTION This member function creates customizers from the given parameterlist and adds them to the customizer vector of the correct type.

RETURNS N/A

END_PROLOG

- GetCSDTImplementation

```
RWCString GetCSDTImplementation() const;
```

Get whether Grid, swath, etc. BEGIN_PROLOG

NAME GetCSDTType

DESCRIPTION This member function is used to get the CSDT that this ESDT is implemented with.

RETURNS String containing CSDT implementation format such as EOSHDF, GRIB, etc.

END_PROLOG

- GetCSDTType

```
RWCString GetCSDTType() const;
```

BEGIN_PROLOG

NAME GetCSDTType

DESCRIPTION This member function is used to get the CSDT that this ESDT is implemented with.

RETURNS String containing CSDT type such as grid, swath, point, etc.

END_PROLOG

- GetMCF

```
EcUtStatus GetMCF(ostream& os) const;
```

Perform termination of this ESDT BEGIN_PROLOG

NAME GetMCF

DESCRIPTION This member function is used to obtain the a stream containing the metadata configuration information.

RETURNS Status indicating success or failure of the request

END_PROLOG

- GetQueryableParameters

```
EcUtStatus GetQueryableParameters(GlParameterList& pl) const;
```

Output: results of validation BEGIN_PROLOG

NAME GetQueryableParameters

DESCRIPTION This member function is used to obtain the metadata attributes that are queryable for this type.

RETURNS Status indicating success or failure of the request

END_PROLOG

- GetType

```
const DsShTypeID* GetType() const;
```

Input: Right-hand side of == BEGIN_PROLOG

NAME GetType

DESCRIPTION This member function returns the data type class of this descriptor implementation.

RETURNS Pointer to data type

GLOBALS None

FILES None

END_PROLOG

- Initialize

```
EcUtStatus Initialize();
```

services that should not be done by just anyone BEGIN_PROLOG

NAME Initialize

DESCRIPTION This member function passes on the initialize request to each of the customizers.

RETURNS Status indicating success or failure of request

END_PROLOG

- operator ==

```
int operator ==(const DsDeDescriptorImp& rhs) const;
```

Input: Created from ODL groups BEGIN_PROLOG

NAME operator==

DESCRIPTION Tests whether two descriptor implementations are equal. This member function is provided so that this class can be used with RWTPtrOrderedVectors. It simply checks for address equality.

RETURNS RWBoolean indication as to whether the two descriptor implementations are equal

END_PROLOG

- operator =

```
DsDeDescriptorImp& operator =(const DsDeDescriptorImp& );
```

copy - NOT IMPLEMENTED

- saveOn

```
void saveOn(ostream& os) const;
```

Get datatype of this implementation BEGIN_PROLOG

NAME saveOn

DESCRIPTION This member function is used for debug purposes. It allows you to print out the contents of this instance while in the debugger.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- SetCustomizers

```
static void SetCustomizers();
```

Input. Used to make customizers BEGIN_PROLOG

NAME SetCustomizers

DESCRIPTION This static member function is used to hold a vector containing the valid customizer strings and associated enums

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- Terminate

```
EcUtStatus Terminate();
```

Perform initialization of this ESDT BEGIN_PROLOG

NAME Terminate

DESCRIPTION This member function passes the terminate request on to each of the customizers.

RETURNS Status indicating success or failure of the request

END_PROLOG

- Validate

```
EcUtStatus Validate(const DsShSciCommandImp& cmd) const;
```

Output: parameters for attributes BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is used to validate the given command.

RETURNS Status indicating whether the command is valid or not

END_PROLOG

- Validate

```
EcUtStatus Validate(istream& is, GlParameterList& plist,
GlStatus& theStatus) const;
```

Input: command to validate BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is used to validate the given metadata stream. It fills in the given parameterlist with the valid metadata. It is used internally when a granule is inserted.

RETURNS Detailed status indicating success or failure of the call

END_PROLOG

- Validate

```
EcUtStatus Validate(istream& is, GlStatus& stat) const;
```

Output: Metadata configuration stream for this type BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is used to validate the given metadata stream.

RETURNS Status indicating success or failure of the call

END_PROLOG

4.3.13.3.10 DsDeDescriptorSet Class

Overview:

DsDeDescriptorSet is the keeper of all descriptors. It is a singleton with two distinct roles to play. One role is for initialization/ termination of all ESDTs in the current sdsrv configuration. The other role is to hand out descriptor implementations. This allows the ability to ensure that only one real descriptor implementation per type exists. This avoids costly parsing of the descriptor file each time a descriptor is created.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myImplementations: RWTPtrOrderedVector<class DsDeDescriptorImp>`

assignment operator - NOT IMPLEMENTED member variables

`ourInstance: DsDeDescriptorSet*`

Descriptors in this configuration

Constructors and Destructor:

`DsDeDescriptorSet();`

`DsDeDescriptorSet(const DsDeDescriptorSet& );`

hidden default constructor the following are defined here and NOT IMPLEMENTED to prevent their use

`~DsDeDescriptorSet();`

Output. Where to write debug info BEGIN_PROLOG

NAME ~DsDeDescriptorSet

DESCRIPTION Destructor for this type. Destroys all of its contents.

RETURNS Status indicating success or failure of request.

END_PROLOG

Operations:

- FindDescriptor

`DsDeDescriptor* FindDescriptor(const DsShTypeID& theType) const;`

static function returns single instance BEGIN_PROLOG

NAME FindDescriptor

DESCRIPTION Looks for a descriptor for the given data type in the current contents.

RETURNS Pointer to the descriptor if found or rwnil if not found

END_PROLOG

- GetDescriptor

`DsDeDescriptorImp* GetDescriptor(const DsShTypeID& theType);`

Input. DataType to look for BEGIN_PROLOG

NAME GetDescriptor

DESCRIPTION Returns a pointer to the desired descriptor. If this type has been previously loaded, then the pointer is found and returned. If the type hasn't been loaded, a descriptor builder

is used to create the implementation. If the type is not a valid type for this dataserver or there is a problem loading the descriptor, an RWxmsg exception is thrown.

RETURNS A pointer to the implementation of the desired type.

END_PROLOG

- Initialize

```
EcUtStatus Initialize();
```

Input. Type of ESDT BEGIN_PROLOG

NAME Initialize

DESCRIPTION Performs initialization on all descriptors in the sdsrv configuration

RETURNS Status indicating success or failure of request.

END_PROLOG

- Instance

```
static DsDeDescriptorSet* Instance();
```

BEGIN_PROLOG

NAME Instance

DESCRIPTION Returns the instance of this singleton.

RETURNS Pointer to the instance, or rwnil if one could not be created.

GLOBALS none

FILES none

END_PROLOG

- operator =

```
DsDeDescriptorSet& operator =(const DsDeDescriptorSet& );
```

copy constructor - NOT IMPLEMENTED

- saveOn

```
void saveOn(ostream& os) const;
```

Terminate all descriptors BEGIN_PROLOG

NAME saveOn

DESCRIPTION This member function is used for debug purposes. It allows you to print out the contents of this instance while in the debugger.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- Terminate

  `EcUtStatus Terminate();`

  Initialize all descriptors BEGIN_PROLOG

  NAME Terminate

  DESCRIPTION Performs termination on all descriptors in the current configuration

  RETURNS Status indicating success or failure of request.

  END_PROLOG

4.3.13.3.11 DsDeDescriptor Class

Overview:

DsDeDescriptor is the class that provides the mechanism for defining and configuring various ESDTs. The kinds of things that are configured include the metadata content, what services are available, what services are advertised, any data dictionary information, and the structural content of an ESDT. These configurable entities are controlled by customizers.

This class reads in the ASCII file that defines the descriptor for a particular type. This is the class that must be concerned with the format of this file. It isolates the parsing of ODL to this class.

This class contains all of the attributes and associated implementation of the descriptor class. The interface is separated from the implementation so that many interfaces can use a single implementation. This is done because descriptor implementations are expensive to create due to the parsing of information.

DsDeDescriptorSet is the keeper of all descriptors. It is a singleton with two distinct roles to play. One role is for initialization/ termination of all ESDTs in the current sdsrv configuration. The other role is to hand out descriptor implementations. This allows the ability to ensure that only one real descriptor implementation per type exists. This avoids costly parsing of the descriptor file each time a descriptor is created.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShDescriptorReal

Attributes:

`myImp: DsDeDescriptorImp*`

Perform termination of this ESDT member variables

`myTypeID: DsShTypeID*`

All of this descriptor's attributes

Constructors and Destructor:

```
DsDeDescriptor(const DsShTypeID& type);
```

```
DsDeDescriptor(const DsDeDescriptor& );
```

This descriptor's datatype the following are declared but NOT IMPLEMENTED to prevent their use

```
virtual  ~DsDeDescriptor();
```

Output. Stream to sent messages to BEGIN_PROLOG

NAME ~DsDeDescriptor

DESCRIPTION Destructor for this class. Deallocates any memory that was allocated.

RETURNS Nothing

END_PROLOG

Operations:

- ConstructedOK

```
EcUtStatus ConstructedOK() const;
```

Get imp type i.e. EOSHDF, GRIB BEGIN_PROLOG

NAME ConstructedOK

DESCRIPTION This member function indicates whether this descriptor was successfully constructed or not.

RETURNS EcUtStatus indicating success or failure

END_PROLOG

- DistConstructedOK

```
virtual EcUtStatus DistConstructedOK() const;
```

I/O: List containing parameters BEGIN_PROLOG

NAME DistConstructedOK

DESCRIPTION This is a distributed member function used by the client to determine whether the descriptor was successfully constructed.

RETURNS Status indicating construction success or failure

END_PROLOG

- DistGetMCF

```
virtual EcUtStatus DistGetMCF(ostream& os) const;
```

Destructor DISTRIBUTED FUNCTIONS BEGIN_PROLOG

NAME DistGetMCF

DESCRIPTION This is a distributed member function used by the client to get the metadata configuration file. It simply passes this request to a local function.

RETURNS Status indicating success or failure of the request

END_PROLOG

- DistGetQueryableParameters

```
virtual EcUtStatus DistGetQueryableParameters(GlParameterList&
pl) const;
```

Output: Status of validation BEGIN_PROLOG

NAME DistGetQueryableParameters

DESCRIPTION This is a distributed member function used by the client to get the metadata attributes in the form of parameters for this type. This function simply passes this request to a local function.

RETURNS Status indicating whether the command is valid or not

END_PROLOG

- DistValidate

```
virtual EcUtStatus DistValidate(istream& is, GlStatus& stat)
const;
```

Output: Metadata configuration file for this type BEGIN_PROLOG

NAME DistValidate

DESCRIPTION This is a distributed member function used by the client to validate metadata. It simply passes this request to a local function.

RETURNS Status indicating success or failure of the call

END_PROLOG

- GetCSDTImplementation

```
RWCString GetCSDTImplementation() const;
```

Get CSDT type i.e. swath, point BEGIN_PROLOG

NAME GetCSDTImplementation

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to obtain the particular implementation of the CSDT (EOSHDF, GRIB, etc.)

RETURNS String containing implementation

END_PROLOG

- GetCSDTType

`RWCString GetCSDTType() const;`

I/O: List containing parameters BEGIN_PROLOG

NAME GetCSDTType

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to obtain the type of CSDT that this ESDT is implemented with.

RETURNS String containing type

END_PROLOG

- GetMCF

`EcUtStatus GetMCF(ostream& os) const;`

LOCAL FUNCTIONS BEGIN_PROLOG

NAME GetMCF

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to obtain the a stream containing the metadata configuration information.

RETURNS Status indicating success or failure of the request

END_PROLOG

- GetQueryableParameters

`EcUtStatus GetQueryableParameters(GlParameterList& pl) const;`

BEGIN_PROLOG

NAME GetQueryableParameters

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to validate the given command.

RETURNS Status indicating whether the command is valid or not

END_PROLOG

- Initialize

`EcUtStatus Initialize();`

services that should not be done by just anyone BEGIN_PROLOG

NAME Initialize

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to perform initialization of this descriptor. The inilization process kicks off advertisements, registers events, exports the data dictionary, etc.

RETURNS Status indicating success or failure of request

END_PROLOG

- operator ==

```
int operator ==(const DsDeDescriptor& rhs) const;
```

Input: Datatype BEGIN_PROLOG

NAME operator==

DESCRIPTION Tests whether two descriptors are equal. This member function is provided so that this class can be used with RWTPtrOrderedVectors. It simply checks for address equality.

RETURNS RWBoolean indication as to whether the two descriptors are equal

END_PROLOG

- operator =

```
DsDeDescriptor& operator =(const DsDeDescriptor& );
```

copy - NOT IMPLEMENTED

- saveOn

```
void saveOn(ostream& os) const;
```

Input: command to validate BEGIN_PROLOG

NAME saveOn

DESCRIPTION This member function is used for debugging purposes. It prints out the contents of this instance.

RETURNS Nothing GLOBALS None

FILES None

END_PROLOG

- Terminate

```
EcUtStatus Terminate();
```

Perform initialization of this ESDT BEGIN_PROLOG

NAME Terminate

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to perform termination of this descriptor. The termination process withdraws advertisements, unregisters events, etc. Termination must be done explicitly meaning that none of the termination functions are automatically done upon destruction of this object or any other objects.

RETURNS Status indicating success or failure of the request

END_PROLOG

- Type

```
Type() const;
```

Input: Right-hand side of == BEGIN_PROLOG

NAME Type

DESCRIPTION Gets the data type of this descriptor

RETURNS A pointer which holds the data type of this descriptor

GLOBALS None

FILES None

END_PROLOG

- Validate

```
EcUtStatus Validate(istream& is, GlStatus& stat) const;
```

Output: Metadata configuration file for this type BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to validate the given metadata stream.

RETURNS Status indicating success or failure of the call

END_PROLOG

- Validate

```
EcUtStatus Validate(istream& is, GlParameterList& plist,
GlStatus& theStatus) const;
```

Output: Status of validation BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to validate the given metadata stream and provide the valid metadata values in the given parameter list.

RETURNS Detailed status indicating success or failure of the call

END_PROLOG

- Validate

```
EcUtStatus Validate(const DsShSciCommandImp& cmd) const;
```

Return datatype of this descriptor BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is implemented by the DsDeDescriptorImp. This is merely a pass-thru to that function. It is used to validate the given command.

RETURNS EcUtStatus indicating whether the command is valid or not

END_PROLOG

4.3.13.3.12 DsDeDictionary Class

Overview:

This file contains the definition of the DsDeDictionaryClass. This class is implemented as a singleton. It provides the ability store a dictionary of attribute types and contents.

DsDeDictionary() Protected contructor. ~DsDeDictionary() Destructor. Instance() Returns a pointer to the single instance of this class. FindAttrContent Performs a look up for a DsDeAttributeContent with a given key FindAttrType Performs a look up for a DsDeAttributeType with a given key FindAttrDefList Performs a look up on a group name and returns a list of DsDeAttributeType Refresh() Refreshes the Hash Dictionaries RefreshType() Refreshes the Dictionary for DsDeAttributeTypes and creates a dictionary of lists that contain types of the same group RefreshContent Refreshes the Dictionary for DsDeAtributeContents IsExternalGroup Returns TRUE if the specified external group name is in the dictionary IsInternalGroup Returns TRUE if the specified internal group name is in the dictionary IsExternalType Returns TRUE if the specified external type name is in the dictionary IsInternalType Returns TRUE if the specified internal type name is in the dictionary ExternalGroupName Returns the external group name for a specified internal group name ExternalTypeName Returns the external type name for a specified internal type name GetTypeDictionary Returns a pointer to the RWPtrHashDictionary that holds attribute types GetContentDictionary Returns a pointer to the RWPtrHashDictionary that holds attribute groups

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myContentDictionary: RWTPtrHashDictionary<class RWCString, class
DsDeAttributeContent>*
```

```
myDefDictionary: RWTPtrHashDictionary<class RWCString, class
RWTPtrSlist<class DsDeAttributeType>>*
```

```
myInternalGroupNameDict: RWTPtrHashDictionary<class RWCString,
class RWCString>*
```

```
myInternalTypeNameDict: RWTPtrHashDictionary<class RWCString,
class RWCString>*
```

```
myTypeDictionary: RWTPtrHashDictionary<class RWCString, class
DsDeAttributeType>*
```

```
ourInstance: DsDeDictionary*
```

Constructors and Destructor:

```
DsDeDictionary();
```

---------- BEGIN_PROLOG

NAME DsDeDictionary

DESCRIPTION Constructor

INPUTS None

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG

```
~DsDeDictionary();
```

constructor ---------- BEGIN_PROLOG

NAME ~DsDeDictionary

DESCRIPTION Destructor

INPUTS None

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG

Operations:

- Add

```
Add();
```

Phase III operation

- ExternalGroupName

  `RWCString ExternalGroupName(const RWCString& group);`

  returns the ODL name ----------
  BEGIN_PROLOG

  NAME ExternalTypeName

  DESCRIPTION Finds the external group name for a given internal group name

  INPUTS RWCString& name

  OUTPUTS None

  RETURNS RWCString- the external group name

  Globals None

  Files None

  END_PROLOG

- ExternalTypeName

  `RWCString ExternalTypeName(const RWCString& name);`

  name is a valid internal name ----------
  BEGIN_PROLOG

  NAME ExternalTypeName

  DESCRIPTION Finds the external type name for a given internal type name

  INPUTS RWCString& name

  OUTPUTS None

  RETURNS RWCString- the external type name

  Globals None

  Files None

  END_PROLOG

- FindAttrContent

  `const DsDeAttributeContent* FindAttrContent(const RWCString& attrKey) const;`

  ---------- BEGIN_PROLOG

  NAME DsMdMetadata::FindAttrContent

  DESCRIPTION Find a named key in the Attribute content dictionary

  INPUTS const RWCString& - the string containing the attribute content key

  OUTPUTS DsMdMCFAttributeContent*& - the structure containing the results from the hash table look-up

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindAttrDefList

```
const RWTPtrSlist<class DsDeAttributeType>*
FindAttrDefList(const RWCString& attrKey) const;
```

search key ---------- BEGIN_PROLOG

NAME DsMdMetadata::FindAttrDefList

DESCRIPTION Find a named key in the Attribute type list dictionary. Returns a list of DsDeAttributeType that have the same group name specified in the key.

INPUTS const RWCString& - the string containing the attribute group key

OUTPUTS DsTDeAttributeType*& - the structure containing the results from the hash table look-up

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindAttrType

```
const DsDeAttributeType* FindAttrType(const RWCString& attrKey)
const;
```

search key ---------- BEGIN_PROLOG

NAME DsMdMetadata::FindAttrType

DESCRIPTION Find a named key in the Attribute type dictionary

INPUTS const RWCString& - the string containing the attribute type key

OUTPUTS DsDeAttributeType*& - the structure containing the results from the hash table look-up

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetContentDictionary

```
const RWTPtrHashDictionary<class RWCString, class
DsDeAttributeContent>* GetContentDictionary();
```

dictionary ---------- BEGIN_PROLOG

NAME GetContentDictionary

DESCRIPTION returns a pointer to the type dictionary

INPUTS None

OUTPUTS None

RETURNS DsTDeContentDictionary* - The pointer to the dictionary

Globals None

Files None

END_PROLOG

- GetTypeDictionary

```
const RWTPtrHashDictionary<class RWCString, class
DsDeAttributeType>* GetTypeDictionary();
```

returns the ODL name ---------- BEGIN_PROLOG

NAME GetTypeDictionary

DESCRIPTION Returns a pointer to the type dictionary

INPUTS None

OUTPUTS None

RETURNS DsTDeTypeDictionary* - The pointer to the dictionary

Globals None

Files None

END_PROLOG

- Instance

```
static DsDeDictionary* Instance();
```

dictionary ---------- BEGIN_PROLOG

NAME Instance

DESCRIPTION Returns a pointer to the single instance of this class

INPUTS None

OUTPUTS None

RETURNS DsDeDictionary*

Globals None

Files None

END_PROLOG

- IsExternalGroup

  `int IsExternalGroup(const RWCString& group);`

  ---------- BEGIN_PROLOG

  NAME IsExternalGroup

  DESCRIPTION Determines if a given RWCString is a valid external group name that exists in the dictionary

  INPUTS RWCString& group

  OUTPUTS None

  RETURNS RWBoolean - True if valid, false if invalid

  Globals None

  Files None

  END_PROLOG

- IsExternalType

  `int IsExternalType(const RWCString& type);`

  name is a valid internal name ---------- BEGIN_PROLOG

  NAME IsExternalType

  DESCRIPTION Determines if a given RWCString is a valid external type name that exists in the dictionary

  INPUTS RWCString& type

  OUTPUTS None

  RETURNS RWBoolean - True if valid, false if invalid

  Globals None

  Files None

  END_PROLOG

- IsInternalGroup

  `int IsInternalGroup(const RWCString& group);`

  name is a valid external name ---------- BEGIN_PROLOG

  NAME IsInternalGroup

  DESCRIPTION Determines if a given RWCString is a valid internal group name that exists in the dictionary

  INPUTS RWCString& group

OUTPUTS None

RETURNS RWBoolean - True if valid, false if invalid

Globals None

Files None

END_PROLOG

- IsInternalType

```
int IsInternalType(const RWCString& type);
```

name is a valid external name ----------
BEGIN_PROLOG

NAME IsInternalType

DESCRIPTION Determines if a given RWCString is a valid Internal type name that exists in the dictionary

INPUTS RWCString& type

OUTPUTS None

RETURNS RWBoolean - True if valid, false if invalid

Globals None

Files None

END_PROLOG

- RefreshContent

```
GlStatus RefreshContent();
```

reloads the Type Dictionary ----------
BEGIN_PROLOG

NAME RefreshContent

DESCRIPTION This functions reloads the Type dictionary from a file specified in the constant DsCDeAttrTypeFile

INPUTS DsCDeAttrContentFile

OUTPUTS None

RETURNS The function returns a GlStatus of SUCCESS if there are tokens for all the attributes of DsDeAttributeType, otherwise FAILURE is returned.

Globals DsDeAttrContentFile - the input file containing data to be inserted.

Files DsCDeAttrContentFile

END_PROLOG

- RefreshType

```
GlStatus RefreshType();
```

destructor ---------- BEGIN_PROLOG

NAME RefreshType

DESCRIPTION This functions reloads the Type and Def dictionary from a file specified in the constant DsCDeAttrTypeFile

INPUTS DsCDeAttrTypeFile

OUTPUTS None

RETURNS The function returns a GlStatus of SUCCESS if there are tokens for all the attributes of DsDeAttributeType, otherwise FAILURE is returned.

GLOBALS

FILES DsDeAttrTypeFile

END_PROLOG

- Refresh

```
GlStatus Refresh();
```

---------- BEGIN_PROLOG

NAME Refresh()

DESCRIPTION Refreshes both Dictionaries.

INPUTS None

OUTPUTS None

RETURNS GlStatus - SUCCESS if both refreshes succeed else FAILURE

Globals None

Files None

END_PROLOG

4.3.13.3.13 DsDeEventCustomizerVector Class

Overview:

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeEventCustomizerVector(istream & from);
```

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.

```
~DsDeCustomizerEventVector();
```

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.

Operations:

4.3.13.3.14 DsDeEventCustomizer Class

Overview:

Provides customization of events.

DsDeEventCustomizer unimplemented default constructor DsDeEventCustomizer(DsDeEventCustomizer&) unimplemented copy constructor DsDeEventCustomizer(GlParamterList&,DsShTypeID) contructor taking a parameter list containg the event name, description, category and status ~DsDeEventCustomizer default destructor operator = unimplemented assignment operator Initialize registers event, gets new EventID, and inserts into event table Terminate unregisters event and removes from event table saveOn outputs content of this class in a readable format

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeCustomizer

Attributes:

```
myEvent: DsClEvent*
```

Constructors and Destructor:

```
DsDeEventCustomizer(const GlParameterList& Parms, const
DsShTypeID& TypeID);
```

```
DsDeEventCustomizer(const DsDeEventCustomizer& );
```

copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeEventCustomizer();
```

The ESDT type. BEGIN_PROLOG

NAME ~DsDeEventCustomizer

DESCRIPTION The destructor for this class

RETURNS none

END_PROLOG

Operations:

- Initialize

```
virtual EcUtStatus Initialize();
```

Destructor BEGIN_PROLOG

NAME Initialize

DESCRIPTION Registers this event, gets an event ID, and inserts the event name and event ID in the event table.

RETURNS EcUtStatus of the operation (OK or FAILED)

END_PROLOG

- operator =

```
DsDeEventCustomizer& operator =(const DsDeEventCustomizer& );
```

assignment operator - NOT IMPLEMENTED

- saveOn

```
virtual void saveOn(ostream& os) const;
```

Unregisters event and removes event from event table BEGIN_PROLOG

NAME saveOn

DESCRIPTION Out puts the contents of this class in a readable format

RETURNS none

END_PROLOG

- Terminate

```
virtual EcUtStatus Terminate();
```

Gets EventID and insert into Event Table BEGIN_PROLOG

NAME Terminate

DESCRIPTION Unregisters the event and removes the event name and event ID// from the event table for this DsShType.

RETURNS EcUtStatus of the operation (OK or FAILED)

END_PROLOG

4.3.13.3.15 DsDeEventTableInfo Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
file: RWFileManager*

os: ostream*
```

Constructors and Destructor:

Operations:

4.3.13.3.16 DsDeEventTable Class

Overview:

Table that holds cross reference of event names and event IDS

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myBtree: RWBTreeOnDisk*`

file that contains the Btree

`myFile: RWFileManager*`

Constructors and Destructor:

`DsDeEventTable(const RWCString& file);`

BEGIN_PROLOG

NAME DsDeEventTable

DESCRIPTION The constructor for this class that takes on a RWCString& for filename

RETURNS none

END_PROLOG

`DsDeEventTable(const DsDeEventTable& );`

copy constructor - NOT IMPLEMENTED

`~DsDeEventTable();`

construct with given file name BEGIN_PROLOG

NAME ~DsDeEventTable

DESCRIPTION The destructor of for this class

RETURNS none

END_PROLOG

Operations:

- FindEventID

`RWCString FindEventID(const RWCString& Name) const;`

key (an Event Name BEGIN_PROLOG

NAME FindEventID

DESCRIPTION Returns the Event ID for a given name

RETURNS RWCString The ID of the Event. A null string if it does not exist.

END_PROLOG

- Insert

`int Insert(const RWCString& Name, const RWCString& EventID);`

BEGIN_PROLOG

NAME Insert

DESCRIPTION Inserts the given Event Name and its Event ID into the table

RETURNS RWBoolean True if successful, FALSE if not

END_PROLOG

- operator =

```
DsDeEventTable& operator =(const DsDeEventTable& );
```

assignment operator - NOT IMPLEMENTED

- Remove

```
void Remove(const RWCString& Name);
```

value (an Event ID BEGIN_PROLOG

NAME Remove

DESCRIPTION Removes an Event Name and its Event ID from the event table

RETURNS none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

key (an Event Name) BEGIN_PROLOG

NAME saveOn

DESCRIPTION Outputs the Event Table to the given output stream in a readable format for debug purposes.

RETURNS none

NOTES A function applyToKeyAndValue is provided by RWBTreeOnDisk that traverses the entire Btree in order. applyToKeyAndValue take two arguments, a pointer to a user defined function and a void pointer. The user defined function operates on each key-value pair. Extra arguments are passed via the viod pointer.

END_PROLOG

4.3.13.3.17 DsDeExpressionRule Class

Overview:

This class contains the functions used by both kinds of expression rules. So far, it simply has a static function to determine whether an operator is a valid one. The intention is that this class never be instantiated which is why the constructors are private.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRule

Attributes:

Constructors and Destructor:

```
DsDeExpressionRule();
```

default constructor - NOT IMPLEMENTED

Operations:

- validOp

```
static int validOp(const RWCString op);
```

BEGIN_PROLOG

NAME validOp

DESCRIPTION This function ensures that the given operator is a valid one.

RETURNS RWBoolean indicating whether operator is valid

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.18 DsDeGroupNode Class

Overview:

This class is used to hold a grouped set of attributes

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeNode

Attributes:

```
myNodes: RWTPtrOrderedVector<class DsDeNode>
```

make group from

Constructors and Destructor:

```
DsDeGroupNode(const RWCString& name);
```

```
DsDeGroupNode(const DsDeGroupNode& );
```

ParameterList containing

```
DsDeGroupNode(const RWCString& name, const GlParameterList& pl);
```

```
~DsDeGroupNode();
```

Output. results/errors BEGIN_PROLOG

NAME ~DsDeGroupNode

DESCRIPTION The destructor for this class.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AddGroup

```
void AddGroup(const RWCString& grp, const GlParameterList& pl);
```

Input. node name BEGIN_PROLOG

NAME AddGroup

DESCRIPTION This method is used to create all of the attributes for this group. If an attribute is not in the given parameterList, it is created from the data dictionary contents.

RETURNS Nothing

GLOBALS None

FILES None

END_PROLOG

- AddParameter

```
void AddParameter(GlParameterList& pl);
```

make parameterlist of atts BEGIN_PROLOG

NAME AddParameter

DESCRIPTION This method creates parameters for each of the nodes in its contents.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- AsParameterList

```
GlParameterList* AsParameterList();
```

BEGIN_PROLOG

NAME AsParameterList

DESCRIPTION This method creates a parameterlist of this groups contents.

RETURNS A newly created parameterlist.

GLOBALS None

FILES None

NOTES The user of this method is responsible for recursive-deleting the return value.

END_PROLOG

- DoSubstitution

```
void DoSubstitution(const GlParameterList& pl);
```

assignment operator BEGIN_PROLOG

NAME DoSubstitution

DESCRIPTION This member function does substitution to all of the nodes in this instances contents.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- Find

```
DsDeNode* Find(const RWCString& name);
```

Input. group name BEGIN_PROLOG

NAME Find

DESCRIPTION This method is used to find a given attribute that is in this instances contents.

RETURNS Pointer to the node or rwnil if not found.

GLOBALS None

FILES None

END_PROLOG

- HasMandatory

```
int HasMandatory(const GlParameterList& outL, GlStatus& stat)
const;
```

Results BEGIN_PROLOG

NAME HasMandatory

DESCRIPTION The destructor for this class.

RETURNS RWBoolean indicates TRUE if parameterlist has all mandatory parameters FALSE otherwise.

GLOBALS None

FILES None

END_PROLOG

- operator =

```
DsDeGroupNode& operator =(const DsDeGroupNode& );
```

copy constructor

- saveOn

```
void saveOn(ostream& os) const;
```

real values to substitute BEGIN_PROLOG

NAME saveOn

DESCRIPTION This member function is used for debugging purposes. It prints out the contents of this instance.

RETURNS Nothing

GLOBALS None

FILES None

END_PROLOG

- ValidateStructure

```
virtual void ValidateStructure(const GlParameter& inL,
GlParameterList& outL, GlStatus& results);
```

Results BEGIN_PROLOG

NAME ValidateStructure

DESCRIPTION This member function is used to validate the structure of the given parameter with respect to the output parameterlist. Valid values are added to the output parameterlist. Error messages are added to the GlStatus object for invalid values.

RETURNS Nothing

GLOBALS None

FILES None

END_PROLOG

- Validate

```
virtual void Validate(const GlParameter& inL, GlParameterList&
outL, GlStatus& results);
```

add parameters of right kind BEGIN_PROLOG

NAME Validate

DESCRIPTION This member function is used to validate the given parameter and if it is valid, add it to the output parameterlist. If the parameter is not valid, an error message is added to the given status object.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.19 DsDeMatchRule Class

Overview:

Distributed Object: No This class has validation criteria that is a series of discrete entries, each of which is valid. In order for something to be valid, the value must match one of the entries in exactly.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRule

Attributes:

```
myDomainValues: RWVector<T>
```

This is a vector that contains an explicit list of valid values for this attribute.

Constructors and Destructor:

```
DsDeMatchRule(istream & from);
```

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.

~DsDeMatchRule();

The destructor for this class has no special implementation.

Operations:

4.3.13.3.20 DsDeMCFtoODL Class

Overview:

This class contains a set of static functions that are used to generate the MCF file in ODL.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

DsDeMCFtoODL();

default constructor - NOT IMPLEMENTED

Operations:

- GetMCFString

static RWCString GetMCFString(const GlStringP& str);

output list from ODL BEGIN_PROLOG

NAME GetMCFString

DESCRIPTION This function is used to create the given GlStringP as the string that will be put into the MCF.

RETURNS MCF-ified RWCString

GLOBALS None

FILES None

END_PROLOG

- IsContainer

```
static int IsContainer(const GlParameterList& pl);
```

parameter used to make MCF string BEGIN_PROLOG

NAME IsContainer

DESCRIPTION This function determines whether the given parameterlist is that of an object that contains other objects otherwise known as a container.

RETURNS RWBoolean indicating plist is a container or not

GLOBALS None

FILES None

END_PROLOG

- MakeContainerODL

```
static void MakeContainerODL(GlParameterList& pl, ostream& os);
```

possible container list BEGIN_PROLOG

NAME MakeContainerODL

DESCRIPTION This function outputs a container objects to the given ostream by using the given parameter list.

RETURNS none

GLOBALS None

FILES None

END_PROLOG

- MakeODL

```
static GlStatus MakeODL(GlParameterList& pl, ostream& os, const
RWCString& kind);
```

output stream of ODL BEGIN_PROLOG

NAME MakeODL

DESCRIPTION This function is used during the creation of the ODL. It is a routine that is called recursively to create the objects and container objects.

RETURNS Status indicating success or failure

GLOBALS None

FILES None

NOTES The status returned is currently always successful. This should be revisited during phase3. END_PROLOG

- ODLtoGl

  `static GlStatus ODLtoGl(istream& , GlParameterList& pl);`

  kind of node - group, object this function is to be replaced with Simon's BEGIN_PROLOG

  NAME ODLtoGl

  DESCRIPTION This is a temporary placeholder for Simon's routine

  RETURNS none

  GLOBALS None

  FILES None

  END_PROLOG

- PlToODL

  `static GlStatus PlToODL(GlParameterList& pl, ostream& os);`

  BEGIN_PROLOG

  NAME PlToODL

  DESCRIPTION This function takes the given parameterlist, converts it to ODL and sends it to the given output stream.

  RETURNS Status indicating success or failure

  GLOBALS None

  FILES None

  NOTES The status returned is currently always successful. This should be revisited during phase3.

  END_PROLOG

4.3.13.3.21 DsDeMetadataCustomizerVector Class

Overview:

This is a container class for the set of metadata customizers for a given datatype. A metadata customizer provides the ability to configure the attributes that a given datatype has. It also provides validation, queryable parameters, and MCF services.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeMetadataCustomizerVector();
```

default constructor BEGIN_PROLOG

NAME DsDeMetadataCustomizerVector

DESCRIPTION Constructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

```
DsDeMetadataCustomizerVector(const
DsDeMetadataCustomizerVector& );
```

copy constructor - NOT IMPLEMENTED

```
~DsDeMetadataCustomizerVector();
```

Output of detailed status destructor BEGIN_PROLOG

NAME ~DsDeMetadataCustomizerVector

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

NOTES The vector of customizers is not owned by this class

END_PROLOG

Operations:

- DoSubstitution

```
void DoSubstitution(const GlParameterList& pl) const;
```

BEGIN_PROLOG

NAME DoSubstitution

DESCRIPTION This method is used to do symbol substitution in each of this vectors customizers.

RETURNS void

GLOBALS None

FILES None

END_PROLOG

- Find

```
DsDeMetadataCustomizer* Find(RWCString mc) const;
```

Output List of metadata parameters BEGIN_PROLOG

NAME Find

DESCRIPTION This method finds a metadata customizer with the given name in this vectors contents. A null pointer is returned if it is not found.

RETURNS DsDeMetadataCustomizer*

GLOBALS None

FILES None

END_PROLOG

- GetMCF

```
EcUtStatus GetMCF(ostream& os) const;
```

BEGIN_PROLOG

NAME GetMCF

DESCRIPTION This method is used to obtain the metadata configuration information for this type and stream it to the provided ostream.

RETURNS Status indicating success or failure of the call

GLOBALS None

FILES None

NOTES This method always return success for now. Additional error handling needs to be added in phase 3.

END_PROLOG

- GetQueryableParameters

```
EcUtStatus GetQueryableParameters(GlParameterList& pl) const;
```

Output Metadata Configuration stream BEGIN_PROLOG

NAME GetQueryableParameters

DESCRIPTION This method is used to obtain the queryable parameters for this type. Each metadata customizer is used to generate the proper parameter type which is added to the given parameterlist.

RETURNS Status indicating success or failure

GLOBALS None

FILES None

NOTES This method always returns success. It seems likely that some error processing will need to be added in phase 3.

END_PROLOG

- HasMandatory

```
int HasMandatory(const GlParameterList& plist, GlStatus& stat)
const;
```

Output of detailed status BEGIN_PROLOG

NAME HasMandatory

DESCRIPTION This method determines if any of the mandatory parameters as defined by the metadata customizers is missing from the given parameterlist. Detailed status of which ones are missing is provided.

RETURNS RWBoolean indicating whether mandatory exists or not

GLOBALS None

FILES None

END_PROLOG

- operator =

```
DsDeMetadataCustomizerVector& operator =(const
DsDeMetadataCustomizerVector& );
```

assignment operator - NOT IMPLEMENTED

- ValidateAttributes

```
void ValidateAttributes(GlParameterList& inPlist,
GlParameterList& validList, GlStatus& stat) const;
```

Name of attribute to find BEGIN_PROLOG

NAME ValidateAttributes

DESCRIPTION This method validates that the metadata attributes are in the configured set for this type, are in the dictionary, etc. It adds the valid ones to the validlist and adds status messages to stat for invalid ones. It is called recursively to handle attributes within attributes.

RETURNS void

GLOBALS None

FILES None

END_PROLOG

- ValidateStructure

```
void ValidateStructure(GlParameterList& inPlist,
GlParameterList& validList, GlStatus& stat) const;
```

Output of detailed status

- Validate

  `EcUtStatus Validate(istream& is, GlParameterList& plist, GlStatus& stat) const;`

  Output status of validation BEGIN_PROLOG

  NAME Validate

  DESCRIPTION This method is used to validate a stream of metadata. It provides a detailed status of what if anything went wrong. This is the method that is intended to be used by external clients.

  RETURNS Status indicating success or failure of the validation Validate

  GLOBALS None

  FILES None

  END_PROLOG

- Validate

  `EcUtStatus Validate(istream& is, GlStatus& stat) const;`

  BEGIN_PROLOG

  NAME Validate

  DESCRIPTION This method is used to validate a stream of metadata. It provides a detailed status of what if anything went wrong. This is the method that is intended to be used by external clients.

  RETURNS Status indicating success or failure of the validation

  GLOBALS None

  FILES None

  END_PROLOG

4.3.13.3.22 DsDeMetadataCustomizer Class

Overview:

Provides the ability to customize the metadata attributes for an ESDT.

This is a container class for the set of metadata customizers for a given datatype. A metadata customizer provides the ability to configure the attributes that a given datatype has. It also provides validation, queryable parameters, and MCF services.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeCustomizer

Attributes:

```
myNode: DsDeNode*
```

Constructors and Destructor:

```
DsDeMetadataCustomizer(const GlParameterList& Parms, const
DsShTypeID& TypeID);
```

```
DsDeMetadataCustomizer(const DsDeMetadataCustomizer& );
```

copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeMetadataCustomizer();
```

The ESDT type. BEGIN_PROLOG

NAME ~DsDeMetadataCustomizer

DESCRIPTION The destructor for this class

RETURNS none

GLOBALS None

FILES None

NOTES The vector of customizers is not owned by this class

END_PROLOG

Operations:

- AsParameterList

```
GlParameterList* AsParameterList() const;
```

debugging aid BEGIN_PROLOG

NAME AsParameterList

DESCRIPTION This function creates a parameterlist from this attribute. The parameterlist consists entirely of GlStringPs.

RETURNS A newly created parameterlist.

GLOBALS None

FILES None

NOTE This is closely tied with the desired MCF. The decision to include attributes or not is based on what is provided in the MCF. The caller of this routine is responsible for doing a recursive delete to free the memory allocated.

END_PROLOG

- DoSubstitution

  `void DoSubstitution(const GlParameterList& pl);`

  output list to add parameters to BEGIN_PROLOG

  NAME DoSubstitution

  DESCRIPTION This member function uses the given parameter list to do substitution for symbolic rules

  RETURNS Nothing

  GLOBALS None

  FILES None

  END_PROLOG

- GetQueryableParameters

  `void GetQueryableParameters(GlParameterList& pl) const;`

  creates this as parameterlist BEGIN_PROLOG

  NAME GetQueryableParameters

  DESCRIPTION This method is used to obtain the queryable parameters for this attribute and add them to the given list.

  RETURNS Nothing

  GLOBALS None

  FILES None

  END_PROLOG

- HasMandatory

  `const int HasMandatory(const GlParameterList& pl, GlStatus& stat) const;`

  name of this attribute BEGIN_PROLOG

  NAME HasMandatory

  DESCRIPTION This function returns an indication of whether this metadata attribute is mandatory or not

  RETURNS RWBoolean indicating whether required is present

  GLOBALS None

  FILES None

  END_PROLOG

- Initialize

  `virtual EcUtStatus Initialize();`

  Destructor BEGIN_PROLOG

NAME Initialize

DESCRIPTION This class doesn't require any special processing upon initialization.

RETURNS EcUtStatus of the operation (always OK)

GLOBALS None

FILES None

END_PROLOG

- Name

```
const RWCString& Name() const;
```

any problems recorded here BEGIN_PROLOG

NAME Name

DESCRIPTION This member function returns the name of this metadata customizer. This equates to the name of a metadata attribute.

RETURNS RWCString with this instances name

GLOBALS None

FILES None

END_PROLOG

- operator =

```
DsDeMetadataCustomizer& operator =(const DsDeMetadataCustomizer&
);
```

assignment operator - NOT IMPLEMENTED

- saveOn

```
virtual void saveOn(ostream& os) const;
```

output for reporting missing BEGIN_PROLOG

NAME saveOn

DESCRIPTION Outputs the contents of this class in a human readable format

RETURNS void

GLOBALS None

FILES None

END_PROLOG

- Terminate

```
virtual EcUtStatus Terminate();
```

does nothing. required by base class BEGIN_PROLOG

NAME Terminate

DESCRIPTION This class doesn't require any special processing upon termination.

RETURNS EcUtStatus of the operation (always OK)

GLOBALS None

FILES None

END_PROLOG

- ValidateStructure

  ```
  void ValidateStructure(const GlParameter& p, GlParameterList&
  valids, GlStatus& stat) const;
  ```

  any problems recorded here

- Validate

  ```
  void Validate(const GlParameter& p, GlParameterList& valids,
  GlStatus& stat) const;
  ```

  does nothing. required by base class table BEGIN_PROLOG

  NAME Validate

  DESCRIPTION The member function validates the given parameter against the rule for this attribute.

  RETURNS Status indicating whether the validation was successful or not.

  GLOBALS None

  FILES None

  NOTE The type of GlParameter (value) that is provided must be of the same type as the rule.

  END_PROLOG

4.3.13.3.23 DsDeNode Class

Overview:

This is the base class for the different attribute types

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myClass: RWCString
```

is this mandatory

`myMandatoryFlag: int`

name of this attribute

`myName: RWCString`

Input. Value to use for class

Constructors and Destructor:

`DsDeNode(const DsDeNode& rhs);`

create from a plist BEGIN_PROLOG

NAME DsDeNode

DESCRIPTION The copy constructor for this class.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`DsDeNode(const GlParameterList& );`

Constructor

`DsDeNode(const RWCString& name);`

BEGIN_PROLOG

NAME DsDeNode

DESCRIPTION The constructor for this class. The given name is the name of the attribute. The contents of the attribute are obtained from the data dictionary.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`virtual  ~DsDeNode();`

Results BEGIN_PROLOG

NAME ~DsDeNode

DESCRIPTION The destructor for this class.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AddParameter

  ```
  virtual void AddParameter(GlParameterList& ) = 0;
  ```

  make parameterlist of MCF atts

- AsParameterList

  ```
  virtual GlParameterList* AsParameterList();
  ```

  debugging aid BEGIN_PROLOG

  NAME AsParameterList

  DESCRIPTION This method creates a parameterlist of the same name as this attribute and inserts parameters for this instances values.

  RETURNS A newly created parameterlist

  GLOBALS None

  FILES None

  NOTES The user of this method is responsible for deleting the parameterList.

  END_PROLOG

- DoSubstitution

  ```
  virtual void DoSubstitution(const GlParameterList& ) = 0;
  ```

  add parameter for this node

- HasMandatory

  ```
  virtual int HasMandatory(const GlParameterList& , GlStatus& )
  const = 0;
  ```

- IsMandatory

  ```
  int IsMandatory() const;
  ```

  test for equality BEGIN_PROLOG

  NAME IsMandatory

  DESCRIPTION This method indicates whether this instance is a mandatory attribute or not.

  RETURNS RWBoolean - TRUE if mandatory, FALSE if optional

  GLOBALS None

  FILES None

  END_PROLOG

- MultiplesOK

  `int MultiplesOK() const;`

  Output. Errors reported here BEGIN_PROLOG

  NAME MultiplesOK

  DESCRIPTION This member function determines whether multiples of this attribute are allowed or not.

  RETURNS RWBoolean indicating whether multiples are allowed.

  GLOBALS None

  FILES None

  END_PROLOG

- Name

  `const RWCString& Name() const;`

  BEGIN_PROLOG

  NAME Name

  DESCRIPTION This member function returns the name of this node.

  RETURNS RWCString containing the name of this node

  GLOBALS None

  FILES None

  END_PROLOG

- operator ==

  `int operator ==(const DsDeNode& );`

  assignment operator BEGIN_PROLOG

  NAME operator==

  DESCRIPTION This member function checks for equality of this instance with the given instance.

  RETURNS RWBoolean indicating whether or not the 2 are equal.

  GLOBALS None

  FILES None

  END_PROLOG

- operator =

  `DsDeNode& operator =(const DsDeNode& rhs);`

  copy constructor BEGIN_PROLOG

  NAME operator=

DESCRIPTION This is the assignment operator. It assigns this instance to the given instance.

RETURNS A reference to this node.

GLOBALS None

FILES None

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

name of this attribute BEGIN_PROLOG

NAME saveOn

DESCRIPTION This member function is used for debugging purposes. It prints out the contents of this instance.

RETURNS Nothing

GLOBALS None

FILES None

END_PROLOG

- SetClass

```
void SetClass(const RWCString& c);
```

Input. Value for mandatory flag BEGIN_PROLOG

NAME SetClass

DESCRIPTION This member function is used to set the class type. The class type is used to indicate whether multiples of this attribute are valid.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- SetMandatoryFlag

```
void SetMandatoryFlag(const int& flag);
```

BEGIN_PROLOG

NAME SetMandatoryFlag

DESCRIPTION This member function is used to set this instances mandatory flag to the given value.

RETURNS Nothing

GLOBALS None

FILES None

END_PROLOG

- ValidateStructure

```
virtual void ValidateStructure(const GlParameter& ,
GlParameterList& , GlStatus& ) = 0;
```

Results

- Validate

```
virtual void Validate(const GlParameter& , GlParameterList& ,
GlStatus& ) = 0;
```

real values for substitution

4.3.13.3.24 DsDeRangeRule Class

Overview:

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRule

Attributes:

myHigh: T

This is the upper limit for the range. Any value that is greater than this value is outside of the valid range.

myLow: T

This is the lower bound for the range. Any value that is less than this value is outside of the valid range.

Constructors and Destructor:

DsDeRangeRule(istream & from);

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.

~DsDeRangeRule();

The destructor removes objects of this type.

Operations:

4.3.13.3.25 DsDeRealExpressionRule ParameterizedClass

Overview:

This template class is a specialization of the RealRule class. It is used to specify an expression for validation. This expression includes an operator and a value. A typical operator is <, >, etc. The value passed in to the IsValid service must be of the same type as this class was instantiated over.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRealRule

Attributes:

`MyOperator: RWCString`

`MyVal: T`

Operator for this expression

Constructors and Destructor:

`DsDeRealExpressionRule(const RWCString op, const T& val);`

constructor

`DsDeRealExpressionRule(const DsDeRealExpressionRule& );`

Returns this rule as a string copy constructor

`virtual ~DsDeRealExpressionRule();`

destructor BEGIN_PROLOG

NAME ~DsDeRealExpressionRule

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AsRuleString

  ```
  virtual RWCString AsRuleString() const;
  ```

  value to validate against expression BEGIN_PROLOG

  NAME AsRuleString

  DESCRIPTION This method string-ifies this rule.

  RETURNS RWCString

  GLOBALS None

  FILES None

  END_PROLOG

- IsValid

  ```
  virtual int IsValid(const void* val) const;
  ```

  value BEGIN_PROLOG

  NAME IsValid

  DESCRIPTION This method determines whether the given value is valid with respect to the expression.

  RETURNS Flag indicating success or failure of validation

  GLOBALS None

  FILES None

  NOTES The value supplied as input must be of the same type as that over which the class was instantiated. The reason that it is of type void* is to allow the function to be templatized and to allow this class to be derived.

  END_PROLOG

- operator =

  ```
  DsDeRealExpressionRule& operator =(const DsDeRealExpressionRule&
  );
  ```

  assignment operator

4.3.13.3.26 DsDeRealMatchRule ParameterizedClass

Overview:

This template class is a specialization of the RealRule class. It is used to specify a set of valid values. These valid values are used to determine whether a given value matches any one in the set. The valid values must be of the same type that this class was instantiated over. In addition, the value passed to the IsValid service must be of this same type.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRealRule

Attributes:

```
MyVals: RWTValOrderedVector
```

Constructors and Destructor:

```
DsDeRealMatchRule();
```

constructor BEGIN_PROLOG

NAME DsDeRealMatchRule

DESCRIPTION Default constructor. The type of the given attributes must be the same as that which the class was instantiated over.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

```
DsDeRealMatchRule(const DsDeRealMatchRule& );
```

Returns this rule as a string copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeRealMatchRule();
```

destructor BEGIN_PROLOG

NAME ~DsDeRealMatchRule

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- Add

```
virtual void Add(const void* elem);
```

BEGIN_PROLOG

NAME Add

DESCRIPTION This method is used to add values to the list of valid values for this rule.

RETURNS void

GLOBALS None

FILES None

NOTES The value supplied as input must be of the same type as that over which the class was instantiated. The reason that it is of type void* is to allow the function to be templatized and to allow this class to be derived.

END_PROLOG

- AsRuleString

```
virtual RWCString AsRuleString() const;
```

value to validate against valid set BEGIN_PROLOG

NAME AsRuleString

DESCRIPTION This method string-ifies this rule.

RETURNS RWCString

GLOBALS None

FILES None

END_PROLOG

- IsValid

```
virtual int IsValid(const void* inVal) const;
```

value to include in the set of valids BEGIN_PROLOG

NAME IsValid

DESCRIPTION This method determines whether the given value is valid with respect to the set of concrete valid values.

RETURNS Flag indicating success or failure of validation

GLOBALS None

FILES None

NOTES The value supplied as input must be of the same type as that over which the class was instantiated. The reason that it is of type void* is to allow the function to be templatized.

END_PROLOG

- operator =

```
DsDeRealMatchRule& operator =(const DsDeRealMatchRule& );
```

assignment operator - NOT IMPLEMENTED

4.3.13.3.27 DsDeRealRangeRule ParameterizedClass

Overview:

This template class is a specialization of the RealRule class. It is used to specify a valid range which includes a minimum value and a maximum value. These values must be of the same type. In addition, the value passed in to the IsValid service must be of the same type as this class was instantiated over.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRealRule

Attributes:

`MyLow:T`

`MyHigh:T`

Constructors and Destructor:

`DsDeRealRangeRule(const T& lo, const T& hi);`

constructor

`DsDeRealRangeRule(const DsDeRealRangeRule& );`

Destructor copy constructor - NOT IMPLEMENTED

`virtual  ~DsDeRealRangeRule();`

Returns this rule as a string BEGIN_PROLOG

NAME ~DsDeRealRangeRule

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AsRuleString

`virtual RWCString AsRuleString() const;`

value to validate against range BEGIN_PROLOG

NAME AsRuleString

DESCRIPTION This method string-ifies this rule.

RETURNS RWCString

GLOBALS None

FILES None

END_PROLOG

- IsValid

```
virtual int IsValid(const void* inVal) const;
```

maximum range value BEGIN_PROLOG

NAME IsValid

DESCRIPTION This method determines whether the given value is valid with respect to the range of this rule.

RETURNS Flag indicating success or failure of validation

GLOBALS None

FILES None

NOTES The value supplied as input must be of the same type as that over which the class was instantiated. The reason that it is of type void* is to allow the function to be templatized.

END_PROLOG

- operator =

```
DsDeRealRangeRule& operator =(const DsDeRealRangeRule& );
```

assignment operator - NOT IMPLEMENTED

4.3.13.3.28 DsDeRealRuleFactory Class

Overview:

This class is a specialization of the RuleFactory class. It makes real rule instances. It is a singleton.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRuleFactory

Attributes:

```
ourInstance: DsDeRealRuleFactory*
```

hidden constructor

Constructors and Destructor:

```
DsDeRealRuleFactory();
```

BEGIN_PROLOG

NAME DsDeRealRuleFactory

DESCRIPTION Default constructor. The constructor is hidden so that only this class can instantiate an instance. This is done through the static Instance function.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

```
DsDeRealRuleFactory(const DsDeRealRuleFactory& );
```

only instance of this class copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeRealRuleFactory();
```

symbolic value BEGIN_PROLOG

NAME ~DsDeRealRuleFactory

DESCRIPTION Destructor for this class. Because this is a singleton, the only time that this method will be called, is when the program is exiting.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- CreateExpressionRule

```
virtual DsDeRule* CreateExpressionRule(enum DsEDeType type,
const RWCString& op, const RWCString& val);
```

matching values BEGIN_PROLOG

NAME CreateExpressionRule

DESCRIPTION This method is used to create a real expression rule. An expression rule is a kind of rule that has an operator and a value representing the right-hand side of an expression.

RETURNS A pointer to a rule. Rule is the base class for all symbolic and real rules.

GLOBALS None

FILES None

NOTE The operator of an expression rule is validated. Because of this, its possible that a rule will not be created.

END_PROLOG

- CreateMatchRule

```
virtual DsDeRule* CreateMatchRule(enum DsEDeType type,
RWTValOrderedVector<class RWCString> valids);
```

maximum of range BEGIN_PROLOG

NAME CreateMatchRule

DESCRIPTION This method is used to create a real match rule. A match rule is a kind of rule that has a discrete set of valid values.

RETURNS A pointer to a rule. Rule is the base class for all symbolic and real rules.

GLOBALS None

FILES None

END_PROLOG

- CreateRangeRule

```
virtual DsDeRule* CreateRangeRule(enum DsEDeType type, const
RWCString& lo, const RWCString& hi);
```

BEGIN_PROLOG

NAME CreateRangeRule

DESCRIPTION This method is used to create a real range rule. A range rule is a kind of rule that has a minimum value and a maximum value.

RETURNS A pointer to a rule. Rule is the base class for all symbolic and real rules.

GLOBALS None

FILES None

END_PROLOG

- Instance

```
static DsDeRuleFactory& Instance();
```

static method used to access this class BEGIN_PROLOG

NAME Instance

DESCRIPTION This member function is the only publically available method of accessing an instance of this class. If this is the first time the instance method is called a new instance of this class is created.

RETURNS A reference to an instance of this class

GLOBALS None

FILES None

END_PROLOG

- operator =

  `DsDeRealRuleFactory& operator =(const DsDeRealRuleFactory& );`

  assignment operator - NOT IMPLEMENTED

4.3.13.3.29 DsDeRealRule Class

Overview:

This class is derived from the Rule class. It is an abstract base class that each specific real rule is derived from. The rule is used to determine if a given value is valid.

This class is a specialization of the RuleFactory class. It makes real rule instances. It is a singleton.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRule

Attributes:

Constructors and Destructor:

`DsDeRealRule();`

default constructor BEGIN_PROLOG

NAME DsDeRealRule

DESCRIPTION Default constructor.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`DsDeRealRule(const DsDeRealRule& );`

Returns the type of this rule BEGIN_PROLOG

NAME DsDeRealRule

DESCRIPTION Copy constructor.

RETURNS N/A

GLOBALS None
FILES None
END_PROLOG

```
virtual  ~DsDeRealRule();
```

Debugging aid BEGIN_PROLOG
NAME ~DsDeRealRule
DESCRIPTION Destructor for this class.
RETURNS N/A
GLOBALS None
FILES None
END_PROLOG

Operations:

- AsRuleString

```
virtual RWCString AsRuleString() const = 0;
```

Destructor

- IsValid

```
virtual int IsValid(const void* ) const = 0;
```

- operator =

```
DsDeRealRule& operator =(const DsDeRealRule& );
```

copy constructor BEGIN_PROLOG
NAME operator=
DESCRIPTION Assignment operator
RETURNS Reference to a real rule
GLOBALS None
FILES None
END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

Value to check validity against BEGIN_PROLOG
NAME saveOn
DESCRIPTION This method outputs the information in this instance to a human readable form.

RETURNS void

GLOBALS None

FILES None

END_PROLOG

- Type

```
Type() const;
```

Returns this rule as a string BEGIN_PROLOG

NAME Type

DESCRIPTION This method indicates the type of this rule. It is supplied to assist in distinguishing between different rule types.

RETURNS Enumerated value indicating that this is a symbolic rule

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.30 DsDeRuleFactory Class

Overview:

This is the base class for all Rule Factories. It provides the interface by which all rules are created. The specialized rule factories create rules of the appropriate type (i.e., symbolic .vs. real).

Note that the caller is responsible for deleting the rule.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeRuleFactory();
```

default constructor BEGIN_PROLOG

NAME DsDeRuleFactory

DESCRIPTION Constructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`DsDeRuleFactory(const DsDeRuleFactory& );`

input string used to create the rule copy constructor BEGIN_PROLOG

NAME DsDeRuleFactory

DESCRIPTION Copy constructor. Doesn't do much of anything because there are no private attributes for this class.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`virtual  ~DsDeRuleFactory();`

destructor BEGIN_PROLOG

NAME ~DsDeRuleFactory

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- CreateExpressionRule

`virtual DsDeRule* CreateExpressionRule(enum DsEDeType , const RWCString& , const RWCString& );`

valid values to match BEGIN_PROLOG

NAME CreateExpressionRule

DESCRIPTION This method defines the interface that derived classes must use to create a expression rule.

RETURNS rwnil

GLOBALS None

FILES None

END_PROLOG

- CreateMatchRule

  ```
  virtual DsDeRule* CreateMatchRule(enum DsEDeType ,
  RWTValOrderedVector<class RWCString> );
  ```

  maximum range value BEGIN_PROLOG

  NAME CreateMatchRule

  DESCRIPTION This method defines the interface that derived classes must use to create a match rule.

  RETURNS rwnil

  GLOBALS None

  FILES None

  END_PROLOG

- CreateRangeRule

  ```
  virtual DsDeRule* CreateRangeRule(enum DsEDeType , const
  RWCString& , const RWCString& );
  ```

  internal factory methods BEGIN_PROLOG

  NAME CreateRangeRule

  DESCRIPTION This method defines the interface that derived classes must use to create a range rule.

  RETURNS rwnil

  GLOBALS None

  FILES None

  END_PROLOG

- MakeRule

  ```
  DsDeRule* MakeRule(DsDeRuleFactory& theFactory, enum DsEDeType
  type, const RWCString& ruleString);
  ```

  BEGIN_PROLOG

  NAME MakeRule

  DESCRIPTION This is the private factory method used to create a rule of the right kind i.e. range, match or expression. It uses the given factory to do the actual rule creations.

  RETURNS A pointer to a newly created rule or rwnil if there was a problem during the rule creation.

  GLOBALS None

  FILES None

  END_PROLOG

- MakeRule

```
DsDeRule* MakeRule(enum DsEDeType type, const RWCString&
ruleString);
```

BEGIN_PROLOG

NAME MakeRule

DESCRIPTION This is the factory method used to create a rule of any kind. It determines what kind of rule is provided and uses the static factory class of the correct type to actually create the rule.

RETURNS A pointer to a newly created rule or rwnil if there was a problem with the rule creation.

GLOBALS None

FILES None NOTES The caller of this routine is responsible for deleting the rule.

END_PROLOG

- operator =

```
DsDeRuleFactory& operator =(const DsDeRuleFactory& );
```

assignment operator BEGIN_PROLOG

NAME operator=

DESCRIPTION The assignment operator doesn't do much of anything because there are no private attributes.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- RuleToVector

```
static RWCString RuleToVector(const RWCString& ruleString,
RWTValOrderedVector<class RWCString>* parms);
```

value for expression BEGIN_PROLOG

NAME RuleToVector

DESCRIPTION This method parses the given string into a vector of strings and determines the kind of rule that this is.

RETURNS A string containing the kind of rule that this is.

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.31 DsDeRule Class

Overview:

This is an abstract base class for all validation rules. It provides the interface by which all rules are created.

NOTES Because the type of the underlying data is not known at compile time, the IsValid method takes a void *. This allows the derived classes to use templates. The technique for implementation of this set of classes was obtained from the Scott Meyers Effective C++ book, p. 154.

This is the base class for all Rule Factories. It provides the interface by which all rules are created. The specialized rule factories create rules of the appropriate type (i.e., symbolic .vs. real).

Note that the caller is responsible for deleting the rule.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DsDeRule();`

BEGIN_PROLOG

NAME DsDeRule

DESCRIPTION Default constructor.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`DsDeRule(const DsDeRule& );`

copy constructor - NOT IMPLEMENTED

`virtual ~DsDeRule();`

Returns type of this rule BEGIN_PROLOG

NAME ~DsDeRule

DESCRIPTION Destructor for this class.
RETURNS N/A
GLOBALS None
FILES None
END_PROLOG

Operations:

- AsRuleString

  ```
  virtual RWCString AsRuleString() const = 0;
  ```

  Right-hand side of ==

- IsValid

  ```
  virtual int IsValid(const void* ) const = 0;
  ```

  default constructor

- operator ==

  ```
  int operator ==(const DsDeRule& rhs) const;
  ```

  destructor BEGIN_PROLOG
  NAME operator==
  DESCRIPTION This method is the test for equality.
  RETURNS N/A
  GLOBALS None
  FILES None
  END_PROLOG

- operator =

  ```
  DsDeRule* operator =(const DsDeRule& );
  ```

  assignment operator - NOT IMPLEMENTED

- saveOn

  ```
  virtual void saveOn(ostream& os) const;
  ```

  value to validate BEGIN_PROLOG
  NAME saveOn
  DESCRIPTION This method outputs the information in this instance to a human readable form.
  RETURNS void
  GLOBALS None

FILES None

END_PROLOG

- Type

```
virtual Type() const = 0;
```

debugging aid

4.3.13.3.32 DsDeServiceCustomizerVector Class

Overview:

This is a container class for the set of service customizers for a given datatype. A service customizer provides the ability to configure the services and associated parameters that an ESDT can perform.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeServiceCustomizerVector();
```

BEGIN_PROLOG

NAME DsDeServiceCustomizerVector

DESCRIPTION Constructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

```
DsDeServiceCustomizerVector(const DsDeServiceCustomizerVector&
);
```

copy constructor - NOT IMPLEMENTED

```
~DsDeServiceCustomizerVector();
```

BEGIN_PROLOG

NAME ~DsDeServiceCustomizerVector

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

NOTES The vector of struct customizers is not owned by this class

END_PROLOG

Operations:

- operator =

```
DsDeServiceCustomizerVector& operator =(const
DsDeServiceCustomizerVector& );
```

operator = - NOT IMPLEMENTED

- Validate

```
EcUtStatus Validate(const DsShSciCommandImp& ) const;
```

BEGIN_PROLOG

NAME Validate

DESCRIPTION

RETURNS

GLOBALS None

FILES None

NOTES This is phase 3 functionality

END_PROLOG

4.3.13.3.33 DsDeServiceCustomizer Class

Overview:

Provides the ability to customize the CSDT type and format that an ESDT is implemented with .

This is a container class for the set of service customizers for a given datatype. A service customizer provides the ability to configure the services and associated parameters that an ESDT can perform.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeCustomizer

Attributes:

```
myAdvertisedFlag: int
```

```
myName: RWCString
```

Constructors and Destructor:

```
DsDeServiceCustomizer(const GlParameterList& pl, const
DsShTypeID& TypeID);
```

```
DsDeServiceCustomizer(const DsDeServiceCustomizer& );
```

unimplemted copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeServiceCustomizer();
```

The ESDT type. BEGIN_PROLOG

NAME ~DsDeServiceCustomizer

DESCRIPTION The destructor for this class

RETURNS none

GLOBALS None

FILES None

END_PROLOG

Operations:

- Initialize

```
virtual EcUtStatus Initialize();
```

Destructor BEGIN_PROLOG

NAME Initialize

DESCRIPTION This function advertises this service if the service is advertiseable.

RETURNS Status indicating success or failure of the call

GLOBALS None

FILES None

END_PROLOG

- IsAdvertised

  ```
  int IsAdvertised() const;
  ```

  prints readable output to an ostream BEGIN_PROLOG

  NAME IsAdvertisedvi DsDe

  DESCRIPTION Returns a flag indicating whether this service should be advertised or not

  RETURNS none

  GLOBALS None

  FILES None

  END_PROLOG

- operator =

  ```
  DsDeServiceCustomizer& operator =(const DsDeServiceCustomizer&
  );
  ```

  assignment operator - NOT IMPLEMENTED

- saveOn

  ```
  virtual void saveOn(ostream& os) const;
  ```

  BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION Outputs the contents of this class in a readable format

  INPUTS ostream& stream that output is to be sent to

  OUTPUTS stream specified by ostream& argument

  RETURNS none

  GLOBALS None

  FILES None

  END_PROLOG

- Terminate

  ```
  virtual EcUtStatus Terminate();
  ```

  Advertise BEGIN_PROLOG

  NAME Terminate

  DESCRIPTION This function does nothing. It is required by the base class.

  RETURNS EcUtStatus of the operation (OK)

  GLOBALS None

  FILES None

  END_PROLOG

- Validate

```
EcUtStatus Validate(const DsShSciCommandImp& ) const;
```

should not be used required by base class table BEGIN_PROLOG

NAME Validate

DESCRIPTION This service validates the given command.

RETURNS Status indicating success or failure of validation.

GLOBALS None

FILES None

NOTES This is phase 3 functionality

END_PROLOG

4.3.13.3.34 DsDeServiceVector Class

Overview:

Distributed Object: No This vector contains DsDeService objects. It is responsible for creating each and for implementing services that operate over the collection of them.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeServiceVector(istream & from);
```

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.

```
~DsDeServiceVector();
```

This destructor invokes the destructor for each of the objects in its contents and then destroys itself.

Operations:

- Advertise

```
RWBoolean Advertise();
```

This service iterates over all of its DsDeService objects and tells each one of them to advertise themselves.

- Find

```
DsDeService* Find(RWCString& svc);
```

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.

- Withdraw

```
RWBoolean Withdraw();
```

This service iterates over all of the DsDeService objects in this vector and invokes the withdraw service.

4.3.13.3.35 DsDeStructCustomizerVector Class

Overview:

This is a container class for the set of struct customizers for a given datatype. A struct customizer provides the ability to configure the CSDT implementation of a datatype.

Distributed Object: No CSDT Structure

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsDeStructCustomizerVector();
```

default constructor BEGIN_PROLOG

NAME DsDeStructCustomizerVector

DESCRIPTION Constructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`DsDeStructCustomizerVector(DsDeStructCustomizerVector& );`

copy constructor - NOT IMPLEMENTED

`~DsDeStructCustomizerVector();`

destructor BEGIN_PROLOG

NAME ~DsDeStructCustomizerVector

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

NOTES The vector of struct customizers is not owned by this class

END_PROLOG

Operations:

- GetCSDTImplementation

  `RWCString GetCSDTImplementation() const;`

  CSDT that this ESDT uses BEGIN_PROLOG

  NAME GetCSDTImplementation

  DESCRIPTION This method returns the name of the implementation method for this ESDT. A typical value for this is EOSHDF.

  RETURNS RWCString

  GLOBALS None

  FILES None

  NOTES The current assumption is that there is only one CSDT per ESDT

  END_PROLOG

- GetCSDTType

  `RWCString GetCSDTType() const;`

  BEGIN_PROLOG

  NAME GetCDSTType

  DESCRIPTION This method returns the name of the CSDT that this ESDT is implemented with. A typical value for this is SWATH.

  RETURNS RWCString

  GLOBALS None

  FILES None

  NOTES The current assumption is that there is only one CSDT per ESDT

END_PROLOG

- operator =

```
DsDeStructCustomizerVector& operator
=(DsDeStructCustomizerVector& );
```

assignment operator - NOT IMPLEMENTED

4.3.13.3.36 DsDeStructCustomizer Class

Overview:

Provides the ability to customize the CSDT type and format that an ESDT is implemented with .

DsDeStructCustomizer unimplemented default constructor DsDeStructCustomizer(DsDeStructCustomizer&) unimplemented copy constructor DsDeStructCustomizer(GlParameterList&,DsShTypeID&) constructor that takes on a CSDT Type and a CSDT format (Implementation) ~DsDeStructCustomizer destructor operator = unimplemented assignment operator Initialize does nothing. Implementation required by base class Terminate does nothing. Implementation required by base class GetCSDTType returns the CSDT Type for this customizer GetCSDTImplementation return the CSDT format (Implementation) saveOn outputs contents of this class in a readable format.

This is a container class for the set of struct customizers for a given datatype. A struct customizer provides the ability to configure the CSDT implementation of a datatype.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeCustomizer

Attributes:

```
myCSDTImplementation: RWCString


myCSDTType: RWCString
```

Constructors and Destructor:

```
DsDeStructCustomizer(const GlParameterList& Parms, const
DsShTypeID& TypeID);
```

```
DsDeStructCustomizer(const DsDeStructCustomizer& );
```

copy constructor - NOT IMPLEMENTED

```
~DsDeStructCustomizer();
```

The ESDT type. BEGIN_PROLOG

NAME ~DsDeStructCustomizer

DESCRIPTION The destructor for this class

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

Operations:

- GetCSDTImplementation

```
RWCString GetCSDTImplementation() const;
```

BEGIN_PROLOG

NAME GetCSDTImplementation

DESCRIPTION Returns the CSDT implementation

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- GetCSDTType

```
RWCString GetCSDTType() const;
```

should not be used required by base class table BEGIN_PROLOG

NAME GetCSDTType

DESCRIPTION Returns the CSDT Type

INPUTS none

OUTPUTS none

RETURNS RWCString

END_PROLOG

- Initialize

```
EcUtStatus Initialize();
```

Destructor BEGIN_PROLOG

NAME Initialize

DESCRIPTION This function does nothing. It is required because it is declared as pure virtual in the base class

INPUTS none

OUTPUTS none

RETURNS EcUtStatus of the operation (OK)

END_PROLOG

- operator =

  `DsDeStructCustomizer& operator =(const DsDeStructCustomizer& );`

  assignment operator - NOT IMPLEMENTED

- saveOn

  `void saveOn(ostream& os) const;`

  BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION Outputs the contents of this class in a readable format

  INPUTS ostream& stream that output is to be sent to

  OUTPUTS stream specified by ostream& argument

  RETURNS none

  END_PROLOG

- Terminate

  `EcUtStatus Terminate();`

  should not be used required by base class BEGIN_PROLOG

  NAME Terminate

  DESCRIPTION This function does nothing. It is required because it is declared as pure virtual in the base class.

  INPUTS none

  OUTPUTS none

  RETURNS EcUtStatus of the operation (OK)

  END_PROLOG

4.3.13.3.37 DsDeSymbolicExpressionRule Class

Overview:

This class is a specialization of the SymbolicRule class. It is used to specify an expression for value validation. A symbolic expression has an operator and a symbolic attribute. A typical operator is <,>, etc.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeSymbolicRule

Attributes:

`myOperator: RWCString`

`myType: const enum DsEDeType`

symbolic name

`myVal: RWCString`

operator, i.e., <, >, etc.

Constructors and Destructor:

`DsDeSymbolicExpressionRule(enum DsEDeType type, const RWCString op, const RWCString val);`

constructor

`DsDeSymbolicExpressionRule(const DsDeSymbolicExpressionRule& );`

type of this rule's data copy constructor - NOT IMPLEMENTED

`virtual ~DsDeSymbolicExpressionRule();`

Returns a rule as a string destructor BEGIN_PROLOG

NAME ~DsDeSymbolicExpressionRule

DESCRIPTION Destructor

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AsRuleString

  ```
  RWCString AsRuleString() const;
  ```

  Real values for symbolic names BEGIN_PROLOG

  NAME AsRuleString

  DESCRIPTION This method is used to "stringify" this rule.

  RETURNS RWCString

  GLOBALS None

  FILES None

  END_PROLOG

- MakeReal

  ```
  virtual void MakeReal(const GlParameterList& pl);
  ```

  debugging aid BEGIN_PROLOG

  NAME MakeReal

  DESCRIPTION This member function converts the symbolic rule into a real rule. This is done by searching the given parameter list for the value.

  RETURNS Nothing

  GLOBALS None

  FILES None

  NOTES If the substitution fails, the factory will create a symbolic rule. This is not a problem, but the validation for this rule will always indicate that the value is invalid.

  END_PROLOG

- operator =

  ```
  DsDeSymbolicExpressionRule& operator =(const
  DsDeSymbolicExpressionRule& );
  ```

  assignment operator - NOT IMPLEMENTED

- saveOn

  ```
  virtual void saveOn(ostream& os) const;
  ```

  symbolic name BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION This method outputs the information in this instance to a human readable form.

  RETURNS void

  GLOBALS None

FILES None

END_PROLOG

4.3.13.3.38 DsDeSymbolicMatchRule Class

Overview:

This class is a specialization of the SymbolicRule class. It is used to specify a set of valid values for an attribute. Any of these values can be symbolic names. The IsValid method ensures that a given value matches at least one of the set in this rule.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeSymbolicRule

Attributes:

```
myType: const enum DsEDeType
```

valid values for this rule

```
myVals: RWTValOrderedVector<class RWCString>
```

Constructors and Destructor:

```
DsDeSymbolicMatchRule(enum DsEDeType type);
```

constructor

```
DsDeSymbolicMatchRule(const DsDeSymbolicMatchRule& );
```

type of this rule's data copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeSymbolicMatchRule();
```

Returns this rule as a string destructor BEGIN_PROLOG

NAME ~DsDeSymbolicMatchRule

DESCRIPTION Destructor

RETURNS None

GLOBALS None

FILES None

END_PROLOG

Operations:

- Add

  ```
  virtual void Add(const RWCString& elem);
  ```

  type of data i.e., int, long BEGIN_PROLOG

  NAME Add

  DESCRIPTION This method is used to add values to the discrete set of valid values that this rule contains. Any or all of these values might be symbolic names.

  RETURNS void

  GLOBALS None

  FILES None

  END_PROLOG

- AsRuleString

  ```
  RWCString AsRuleString() const;
  ```

  Real values for symbolic names BEGIN_PROLOG

  NAME AsRuleString

  DESCRIPTION This method is used to "stringify" this rule.

  RETURNS RWCString

  GLOBALS None

  FILES None

  END_PROLOG

- MakeReal

  ```
  virtual void MakeReal(const GlParameterList& pl);
  ```

  debugging aid BEGIN_PROLOG

  NAME MakeReal

  DESCRIPTION This member function converts the symbolic rule into a real rule. This is done by searching the given parameter list for the value.

  RETURNS Nothing

  GLOBALS None

  FILES None

  NOTES If the substitution fails, the factory will create a symbolic rule. This is not a problem, but the validation for this rule will always indicate that the value is invalid.

  END_PROLOG

- operator =

```
DsDeSymbolicMatchRule& operator =(const DsDeSymbolicMatchRule&
);
```

assignement operator - NOT IMPLEMENTED

- saveOn

```
virtual void saveOn(ostream& os) const;
```

valid value for this rule BEGIN_PROLOG

NAME saveOn

DESCRIPTION This method outputs the information in this instance to a human readable form.

RETURNS void

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.39 DsDeSymbolicRangeRule Class

Overview:

This class is a specialization of the SymbolicRule class. It is used to specify a valid range which includes a minimum value and a maximum value. Either or both of these values can be symbolic attributes.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeSymbolicRule

Attributes:

```
myHi: RWCString
```

minimum value string

```
myLow: RWCString
```

type of this rule's data

```
myType: enum DsEDeType
```

Constructors and Destructor:

```
DsDeSymbolicRangeRule(enum DsEDeType type, const RWCString& lo,
const RWCString& hi);
```

constructor

```
DsDeSymbolicRangeRule(const DsDeSymbolicRangeRule& );
```

maximum value string copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeSymbolicRangeRule();
```

Returns a rule as a string destructor BEGIN_PROLOG

NAME DsDeSymbolicRangeRule

DESCRIPTION Destructor.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AsRuleString

```
RWCString AsRuleString() const;
```

Real values for symbolic names BEGIN_PROLOG

NAME AsRuleString

DESCRIPTION This method is used to "stringify" this rule.

RETURNS RWCString

GLOBALS None

FILES None

END_PROLOG

- MakeReal

```
virtual void MakeReal(const GlParameterList& pl);
```

Debugging aid BEGIN_PROLOG

NAME MakeReal

DESCRIPTION This member function converts the symbolic rule into a real rule. This is done by searching the given parameter list for the value.

RETURNS

GLOBALS None

FILES None

NOTES If the substitution fails, the factory will create a symbolic rule. This is not a problem, but the validation for this rule will always indicate that the value is invalid.

END_PROLOG

- operator =

```
DsDeSymbolicRangeRule& operator =(const DsDeSymbolicRangeRule&
);
```

assignment operator - NOT IMPLEMENTED

- saveOn

```
virtual void saveOn(ostream& os) const;
```

maximum value of range BEGIN_PROLOG

NAME saveOn

DESCRIPTION This method outputs the information in this instance to a human readable form.

RETURNS void

GLOBALS None

FILES None

END_PROLOG

4.3.13.3.40 DsDeSymbolicRuleFactory Class

Overview:

This class is a specialization of the RuleFactory class. It makes symbolic rule instances. It is a singleton.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRuleFactory

Attributes:

```
ourInstance: DsDeSymbolicRuleFactory*
```

hidden constructor

Constructors and Destructor:

```
DsDeSymbolicRuleFactory();
```

BEGIN_PROLOG

NAME DsDeSymbolicRuleFactory

DESCRIPTION Default constructor. The constructor is hidden so that only this class can instantiate an instance. This is done through the static Instance function.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

```
DsDeSymbolicRuleFactory(const DsDeSymbolicRuleFactory& );
```

copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeSymbolicRuleFactory();
```

symbolic value destructor BEGIN_PROLOG

NAME ~DsDeSymbolicRuleFactory

DESCRIPTION Destructor for this class. Because this is a singleton, the only time that this method will be called is when the program is exiting.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- CreateExpressionRule

```
virtual DsDeRule* CreateExpressionRule(enum DsEDeType type,
const RWCString& op, const RWCString& val);
```

matching values BEGIN_PROLOG

NAME CreateExpressionRule

DESCRIPTION This method is used to create a symbolic expression rule. An expression rule is a kind of rule that has an operator and a value representing the right-hand side of an expression. The fact that it is a symbolic rule means that value substitution must be done before the rule can be applied.

RETURNS A pointer to a rule. Rule is the base class for all symbolic and real rules.

GLOBALS None

FILES None

NOTE The operator of an expression rule is validated. Because of this, its possible that a rule will not be created.

END_PROLOG

- CreateMatchRule

  ```
  virtual DsDeRule* CreateMatchRule(enum DsEDeType type,
  RWTValOrderedVector<class RWCString> valids);
  ```

  maximum of range BEGIN_PROLOG

  NAME CreateMatchRule

  DESCRIPTION This method is used to create a symbolic match rule. A match rule is a kind of rule that has a discrete set of valid values. The fact that it is a symbolic rule means that some value substitution must be done before the rule can be applied. Any or all of the input strings can be symbolic names.

  RETURNS A pointer to a rule. Rule is the base class for all symbolic and real rules.

  GLOBALS None

  FILES None

  END_PROLOG

- CreateRangeRule

  ```
  virtual DsDeRule* CreateRangeRule(enum DsEDeType type, const
  RWCString& lo, const RWCString& hi);
  ```

  BEGIN_PROLOG

  NAME CreateRangeRule

  DESCRIPTION This method is used to create a symbolic range rule. A range rule is a kind of rule that has a minimum value and a maximum value. The fact that it is a symbolic rule means that some value substitution must be done before the rule can be applied. Either or both of the input strings can be symbolic names.

  RETURNS A pointer to a rule. Rule is the base class for all symbolic and real rules.

  GLOBALS None

  FILES None

  END_PROLOG

- Instance

  ```
  static DsDeRuleFactory& Instance();
  ```

  static method used to access this class BEGIN_PROLOG

  NAME Instance

  DESCRIPTION This member function is the only publically available method of accessing an instance of this class. If this is the first time the instance method is called, a new instance of this class is created.

  RETURNS A reference to an instance of this class

  GLOBALS None

  FILES None

END_PROLOG

- operator =

```
DsDeSymbolicRuleFactory& operator =(const
DsDeSymbolicRuleFactory& );
```

assignment operator - NOT IMPLEMENTED

4.3.13.3.41 DsDeSymbolicRule Class

Overview:

This class is derived from the Rule class. It is an abstract base class that each specific symbolic rule is derived from. A symbolic rule provides the ability to express a validation rule in a symbolic form. The rule is used to determine if a given value is valid. If the symbolic rule has not been made into a real rule, no value is valid.

This class is a specialization of the RuleFactory class. It makes symbolic rule instances. It is a singleton.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDeRule

Attributes:

```
myRealRule: DsDeRule*
```

Constructors and Destructor:

```
DsDeSymbolicRule();
```

default constructor

```
DsDeSymbolicRule(const DsDeSymbolicRule& );
```

pointer to this symbolic rules real rule copy constructor - NOT IMPLEMENTED

```
virtual  ~DsDeSymbolicRule();
```

Returns this rule as a string BEGIN_PROLOG

NAME ~DsDeSymbolicRule

DESCRIPTION Destructor for this class.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AsRuleString

```
virtual RWCString AsRuleString() const = 0;
```

Debugging function

- DoSubstitution

```
static void DoSubstitution(RWCString* symString, const
GlParameterList& realVals);
```

Destructor BEGIN_PROLOG

NAME DoSubstitution

DESCRIPTION This method is used to substitute real values for symbolic values. It is a static method used by derived classes when turning symbolic rules into real rules.

RETURNS Nothing

GLOBALS None

FILES None

END_PROLOG

- IsValid

```
virtual int IsValid(const void* inVal) const;
```

What kind of rule is this BEGIN_PROLOG

NAME IsValid

DESCRIPTION This method applies the rule to a given value. If this rule has not been made into a real rule via substitution, then the value is not valid.

RETURNS RWBoolean indicating whether given value is valid

GLOBALS None

FILES None

NOTE The type of this argument is void* so that validation can be done generically. The type of the given value must be the same as the type that the underlying rule was instantiated over.

END_PROLOG

- MakeReal

```
virtual void MakeReal(const GlParameterList& ) = 0;
```

- operator =

  ```
  DsDeSymbolicRule& operator =(const DsDeSymbolicRule& );
  ```

  assignment operator - NOT IMPLEMENTED

- saveOn

  ```
  virtual void saveOn(ostream& os) const = 0;
  ```

  Value to check validity against

- Type

  ```
  Type() const;
  ```

  Plist has values for symbolics BEGIN_PROLOG

  NAME Type

  DESCRIPTION This method indicates the type of this rule. It is supplied to assist in distinguishing between different rule types.

  RETURNS Enumerated value indicating that this is a symbolic rule

  GLOBALS None

  FILES None

  END_PROLOG

4.3.14 dm Class Category

4.3.14.1 Overview

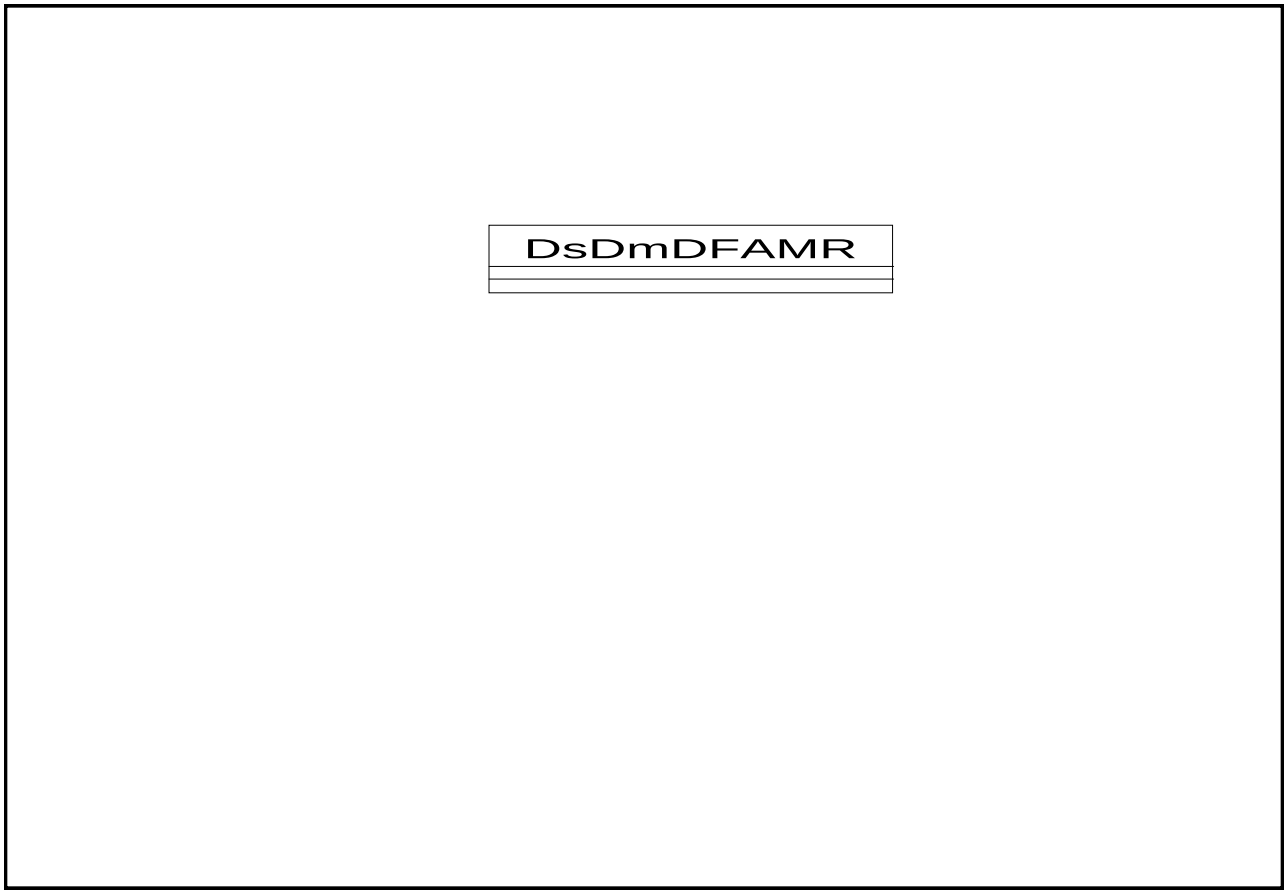


Figure 23 Main

4.3.14.2 dm Classes

4.3.14.2.1 DsDmDFAMR Class

Overview:

Distrubuted Object: No This class is an abastraction of the products derived from DFA/MR instruments.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsDmDFAMR();

~DsDmDFAMR();
```

Operations:

4.3.15 er Class Category

4.3.15.1 Overview

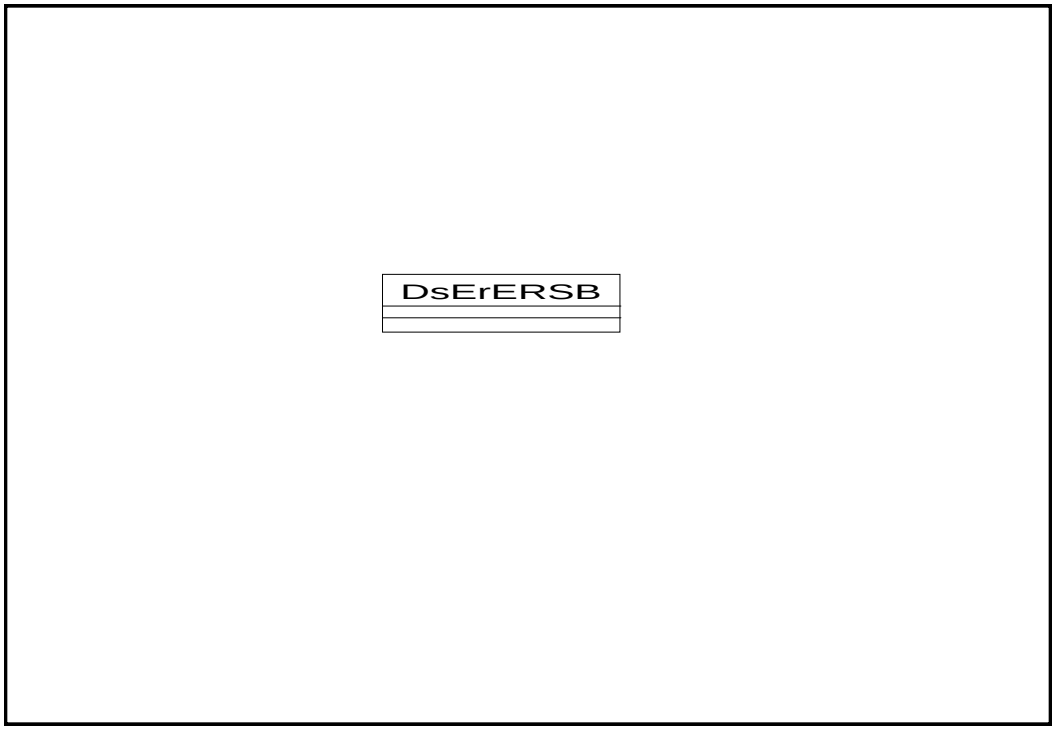


Figure 24 Main

4.3.15.2 er Classes

4.3.15.2.1 DsErERSB Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the instruments on the ERS satellite.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsErERSB();

~DsErERSB();
```

Operations:

4.3.16 et Class Category

4.3.16.1 Overview

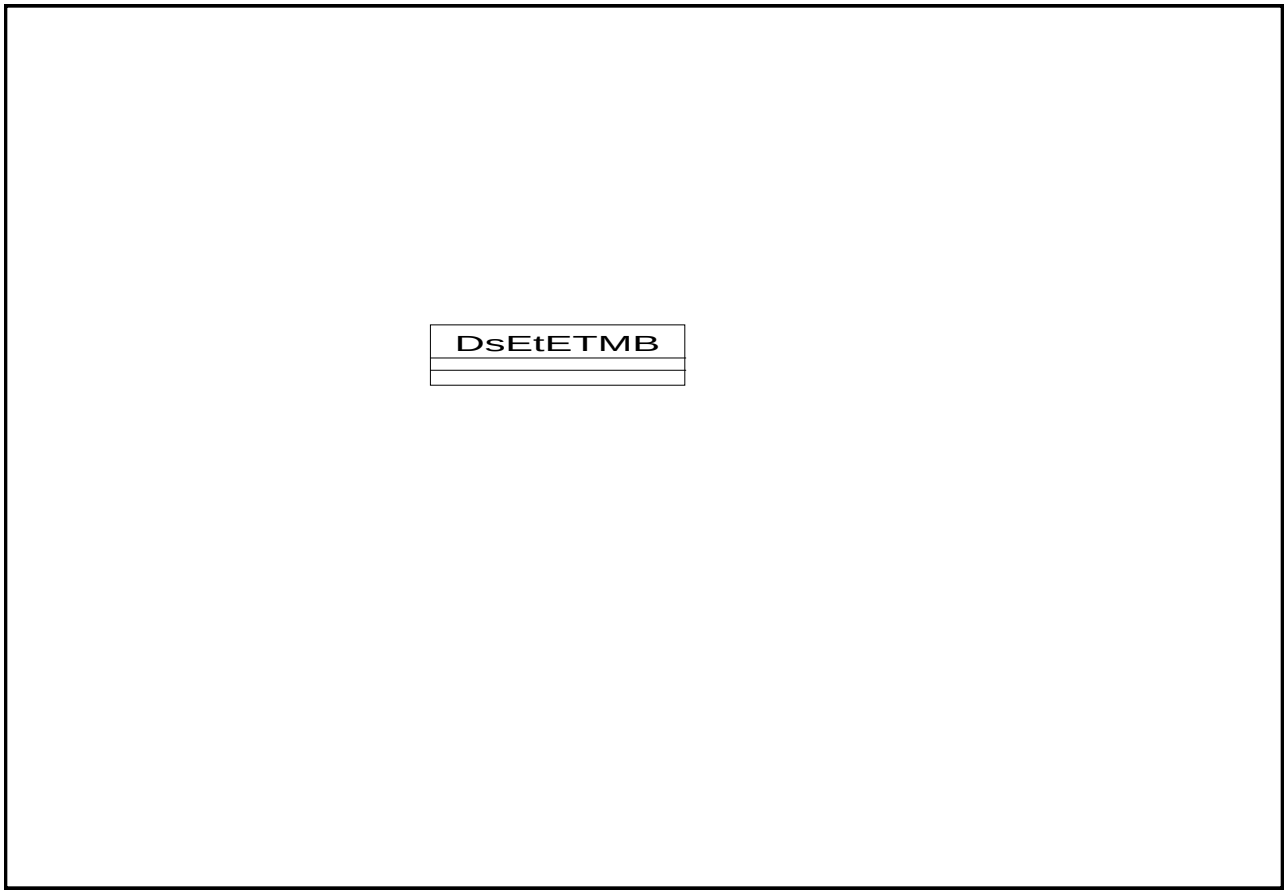


Figure 25 Main

4.3.16.2 et Classes

4.3.16.2.1 DsEtETMB Class

Overview:

DIstriubted Object: No This class is an abstraction of the products derived from the ETM Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
~DsEtETMB();
```

Operations:

- DsErETMB

```
DsErETMB();
```

4.3.17 ge Class Category

4.3.17.1 Overview

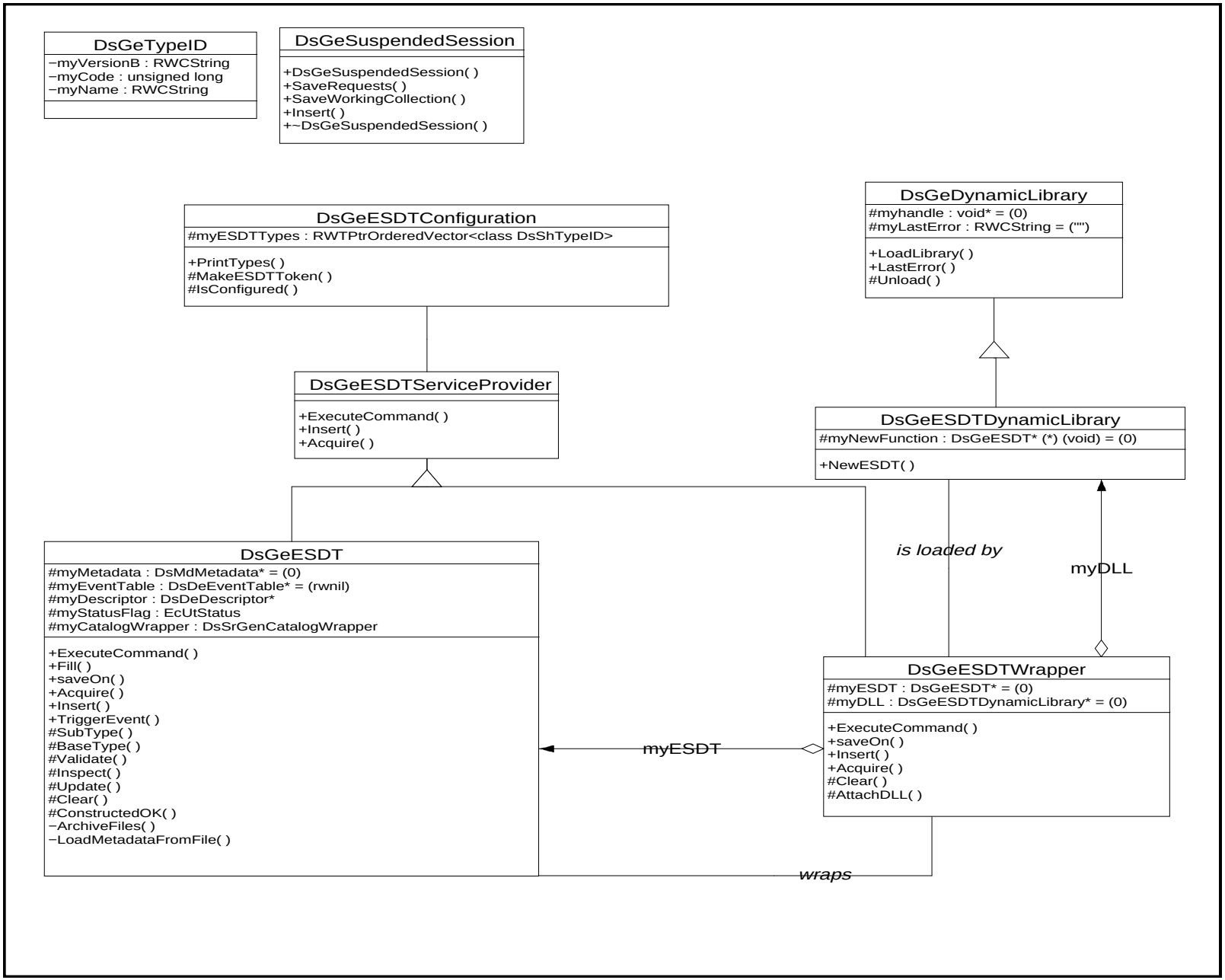


Figure 20 DsGe1

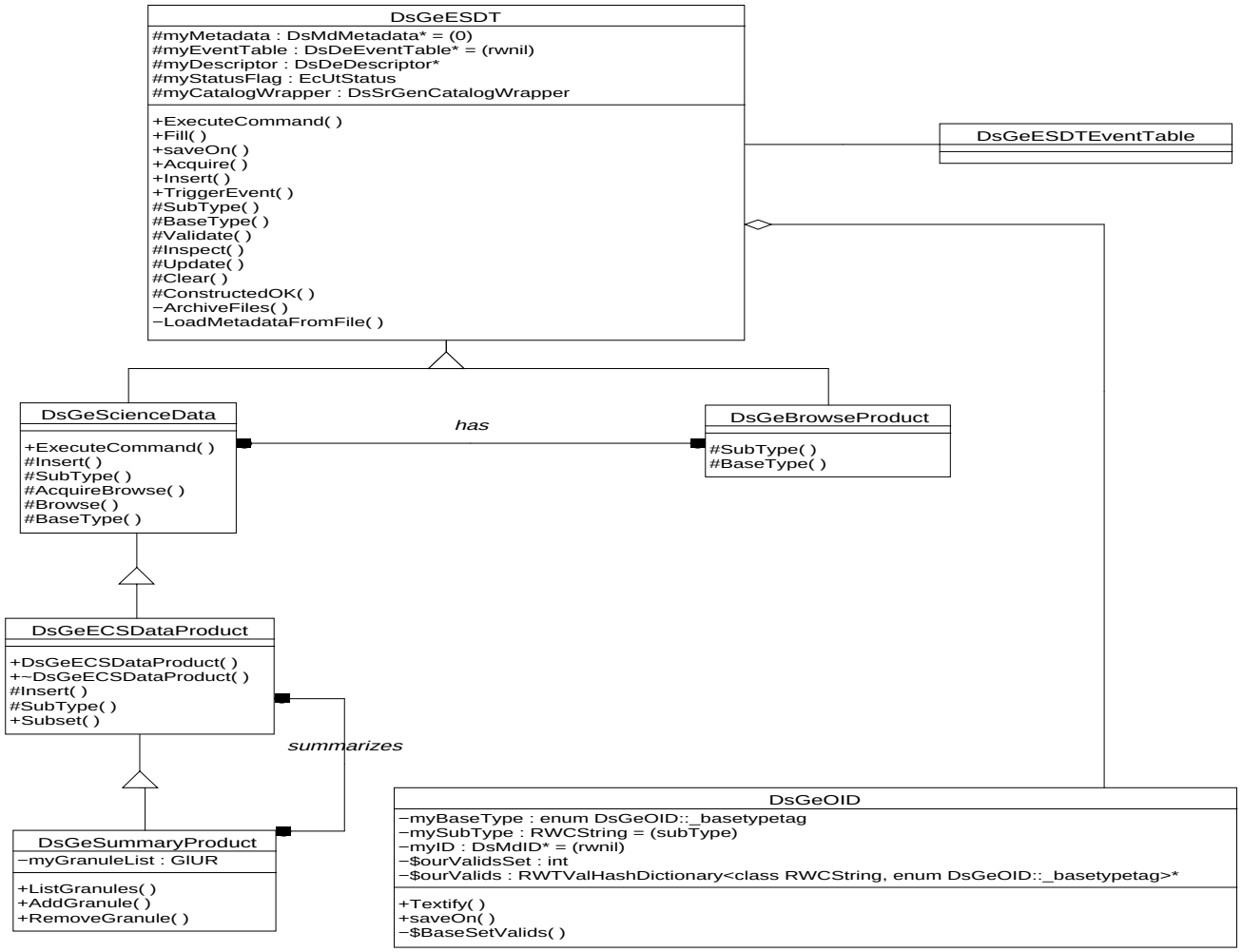


Figure 20 DsGe2

4.3.17.2 ge Classes

4.3.17.2.1 DsGeBrowseProduct Class

Overview:

This file becomes a dynamic library that implements the Browse data type. So, the "standard" entry point DsGeNewESDT is provided

inherit from the public class - provides basic ESDT functionality

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeESDT

Attributes:

Constructors and Destructor:

```
DsGeBrowseProduct();
```

```
virtual  ~DsGeBrowseProduct();
```

default constructor BEGIN_PROLOG

NAME ~DsGeBrowseProduct

DESCRIPTION Destructor for this class

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

NOTES

END_PROLOG

Operations:

- BaseType

```
virtual enum DsGeOID::_basetypetag BaseType() const;
```

Returns subtype of this instance BEGIN_PROLOG

NAME BaseType

DESCRIPTION This function returns the basetype name of this ESDT instance.

OUTPUTS None

RETURNS DsEGeBaseType - enum defined by GeOID class.

GLOBALS None

FILES None

NOTES

END_PROLOG

- SubType

```
virtual RWCString SubType() const;
```

BEGIN_PROLOG

NAME SubType

DESCRIPTION This function returns the datatype name of this ESDT instance.

OUTPUTS None

RETURNS An RWCString with the name of this ESDT instance.

GLOBALS None

FILES None

NOTES

END_PROLOG

4.3.17.2.2 DsGeDynamicLibrary Class

Overview:

This object is used to perform runtime loading of libraries. It uses DL library routines dlopen, dlclose, dlerror, and dlsym to accomplish this task.

class that implements the dynamic library loading

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myhandle: void*
```

Set last DLL error

```
myLastError: RWCString
```

Constructors and Destructor:

```
DsGeDynamicLibrary();
```

```
virtual  ~DsGeDynamicLibrary();
```

constructor BEGIN_PROLOG

NAME ~DsGeDynamicLibrary

DESCRIPTION The destructor unloads the dynamically loaded library.

RETURNS N/A

GLOBALS N/A

FILES N/A

END_PROLOG

Operations:

- GetSymbol

```
virtual void* GetSymbol(const RWCString& fnname) const;
```

Input. Name of the library to load BEGIN_PROLOG

NAME GetSymbol

DESCRIPTION This function looks up the given symbol's address in the DLL and returns it.

INPUTS symbol - String containing the name to look up

OUTPUTS None

RETURNS A void pointer to the symbol

GLOBALS None

FILES None

END_PROLOG

- LastError

```
const RWCString LastError() const;
```

Input. Name of symbol to look up BEGIN_PROLOG

NAME LastError

DESCRIPTION This function returns a string describing the last error that occurred in a call to this object.

INPUTS None

OUTPUTS None

RETURNS A string describing the last error that occurred in a call to this object

GLOBALS

FILES

END_PROLOG

- LoadLibrary

```
virtual int LoadLibrary(const RWCString& name);
```

destructor BEGIN_PROLOG

NAME LoadLibrary

DESCRIPTION This function loads the given library into memory.

INPUTS Name of the file containing the dynamically loadable library.

OUTPUTS

RETURNS True indicating the library was successfully loaded or False indicating there was some problem loading the library.

GLOBALS None

FILES The name of the DLL which is provided.

END_PROLOG

- SetLastError

```
virtual void SetLastError();
```

Unload the dynamic library BEGIN_PROLOG

NAME SetLastError

DESCRIPTION This function sets the variable that maintains the error state of this object. It uses the dlerror system library routine.

INPUTS None

OUTPUTS None

RETURNS None

GLOBALS

FILES

END_PROLOG

- Unload

```
virtual void Unload();
```

BEGIN_PROLOG

NAME Unload

DESCRIPTION

This function unloads the DLL which is currently loaded

INPUTS None

OUTPUTS None

RETURNS None

GLOBALS

FILES

END_PROLOG

4.3.17.2.3 DsGeECSDataProduct Class

Overview:

The following classes are defined in this file: DsGeECSDataProduct Specialized ESDT for ECS data

The following const variables are defined in this file: none

The following defines are defined in this file: DsGeECSDataProductH ensures file isn't read twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following public functions are defined in this file: DsGeECSDataProduct() default constructor ~DsGeECSDataProduct() default destructor

The following protected functions are defined in this file: virtual Insert(const DsSrCommand &,); Insert an ESDT into SDSRV GlParameterList &

virtual RWCString SubType() const;

inherit from the public ESDT class, which provides basic ESDT functionality Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeScienceData

Attributes:

Constructors and Destructor:

```
DsGeECSDataProduct(const DsShTypeID& type);
```

```
virtual  ~DsGeECSDataProduct();
```

BEGIN_PROLOG

NAME ~DsGeECSDataProduct

DESCRIPTION This is the default destructor for the DsGeECSDataProduct datatype. It simply calls the parent class destructor.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- Insert

```
virtual EcUtStatus Insert(const GlParameterList& commandParams,
GlParameterList& insertResults);
```

BEGIN_PROLOG

NAME Insert

DESCRIPTION Generic insert command for ECS Data Product based ESDT's. This is provided for those ECS Data products that do not require a data specific insert service. This insert command is responsible for handling relationships between DsNsScienceSoftwareArchivePackage & ECS Data products.

INPUTS const DsSrCommand& - reference to command object, contains name of ESDT service to execute and parameterlist of associated command parameters.

OUTPUTS GlParameterList& - reference to parameter list into which results of command execution are placed (contents dependent on type of command) RETURNS EcUtStatus - Completion status of command

GLOBALS None

FILES None

NOTES None

END_PROLOG

- Subset

```
GlUR Subset(GlParameterList PList);
```

- SubType

```
virtual RWCString SubType() const = 0;
```

4.3.17.2.4 DsGeESDTConfiguration Class

Overview:

This class is used to determine ESDT configuration information. It is derived from the DsCnConfiguration class which provides the basic opening, closing and getting token services. This class adds the services that are specific to ESDTs including parsing for specific well-defined tokens, determining the set of ESDTs currently available, and obtaining the DLL, descriptor, and event table handles.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsCnConfiguration

Attributes:

```
myESDTTypes: RWTPtrOrderedVector<class DsShTypeID>
```

Constructors and Destructor:

```
DsGeESDTConfiguration(const RWCString& fname);
```

```
DsGeESDTConfiguration();
```

Input. Name of configuration file

```
virtual  ~DsGeESDTConfiguration();
```

default constructor BEGIN_PROLOG

NAME ~DsGeESDTConfiguration

DESCRIPTION The destructor frees all previously allocated memory.

INPUTS None

OUTPUTS N/A

RETURNS N/A

GLOBALS N/A

FILES N/A

END_PROLOG

Operations:

- GetAllDataTypes

  ```
  const RWTPtrOrderedVector<class DsShTypeID> GetAllDataTypes() const;
  ```

  destructor BEGIN_PROLOG

  NAME GetAllDataTypes

  DESCRIPTION This member function returns a vector of all of the available ESDTs for this ESDT configuration.

  INPUTS None

  OUTPUTS None

  RETURNS A vector of all of the available ESDTs

  GLOBALS None

  FILES None

  END_PROLOG

- GetDefinitionFileForType

  ```
  RWCString GetDefinitionFileForType(const DsShTypeID& name) const;
  ```

  get list of available ESDTs BEGIN_PROLOG

  NAME GetDefinitionFileForType

  DESCRIPTION This function looks up the filename/handle of the descriptor for the given type.

  INPUTS name - Name of the datatype to get the descriptor of

  OUTPUTS None

  RETURNS The filename/handle of the descriptor for the type.

  GLOBALS N/A

  FILES N/A

  END_PROLOG

- GetEventTableForType

  ```
  RWCString GetEventTableForType(const DsShTypeID& name) const;
  ```

  Input. The datatype identifier BEGIN_PROLOG

NAME GetEventTableForType

DESCRIPTION This member function returns the filename/handle for the event table for the given datatype. The event table is used by the ESDTs to determine what events to trigger when subscribable events occur.

INPUTS name - Name of the datatype to get the event table for

OUTPUTS None

RETURNS A string with the filename/handle for the event table for the given datatype

GLOBALS N/A

FILES N/A

END_PROLOG

- GetLibraryForType

```
RWCString GetLibraryForType(const DsShTypeID& name) const;
```

Input. The datatype identifier BEGIN_PROLOG

NAME GetLibraryForType

DESCRIPTION This function determines the name of the dynamically loaded library (DLL) for the given datatype.

INPUTS name - Name of the datatype to get the DLL for

OUTPUTS None

RETURNS A string with the name/handle of the DLL for the given datatype

GLOBALS N/A

FILES N/A

END_PROLOG

- IsConfigured

```
int IsConfigured(const DsShTypeID& name) const;
```

Input. Use to make ESDT token BEGIN_PROLOG

NAME IsConfigured

DESCRIPTION This function determines whether the given datatype is configured in the current configuration. If the datatype is not configured many of the member functions that involve getting information for a specific datatype will log an error and return an empty value.

INPUTS name - Datatype name to search for in the configuration

OUTPUTS None

RETURNS TRUE if the given type is configured, FALSE if it is not

GLOBALS N/A

FILES N/A

END_PROLOG

- MakeESDTToken

  `RWCString MakeESDTToken(const DsShTypeID& name) const;`

  available ESDTs BEGIN_PROLOG

  NAME MakeESDTToken

  DESCRIPTION Create an ESDT Token from the given string.

  INPUTS name - A string used to create an ESDT Token

  OUTPUTS None

  RETURNS An ESDT token created using the given string

  GLOBALS N/A

  FILES N/A

  END_PROLOG

- PrintTypes

  `void PrintTypes(ostream& os);`

  Input. The datatype identifier BEGIN_PROLOG

  NAME PrintTypes

  DESCRIPTION This member function is used for debugging purposes. It outputs all of the configured ESDTs in this configuration.

  INPUTS None

  OUTPUTS os - Stream to send output to

  RETURNS N/A

  GLOBALS N/A

  FILES N/A

  END_PROLOG

- SetConfiguredTypes

  `void SetConfiguredTypes();`

  name of ESDT BEGIN_PROLOG

  NAME SetConfiguredTypes

  DESCRIPTION This method identifies the ESDTs that are available in the current configuration.

  RETURNS N/A

  GLOBALS N/A

  FILES N/A

  END_PROLOG

4.3.17.2.5 DsGeESDTDynamicLibrary Class

Overview:

This class performs runtime loading of ESDT libraries for a data type. It encapsulates the public functionality necessary for ESDT DLLs.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeDynamicLibrary

Attributes:

`myNewFunction: DsGeESDT* (*) (void)`

hidden default constructor

Constructors and Destructor:

`DsGeESDTDynamicLibrary(const RWCString& libname);`

`DsGeESDTDynamicLibrary();`

BEGIN_PROLOG

NAME DsGeESDTDynamicLibrary

DESCRIPTION This is the default constructor for this class. It is hidden from public view.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`~DsGeESDTDynamicLibrary();`

Input. Name of ESDT library with path BEGIN_PROLOG

NAME ~DsGeESDTDynamicLibrary

DESCRIPTION The destructor ensures that the NewESDT function is no longer accessible. The base class destructor unloads the library.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS N/A

FILES

END_PROLOG

Operations:

- NewESDT

```
DsGeESDT* NewESDT();
```

destructor BEGIN_PROLOG

NAME NewESDT

DESCRIPTION This functions calls the function that allocates a new ESDT of the correct type and returns a pointer to it. This ESDT must be freed by the caller of the NewESDT function.

INPUTS None

OUTPUTS None

RETURNS A base class pointer that points to an instance of the correct type.

GLOBALS N/A

FILES None

END_PROLOG

4.3.17.2.6 DsGeESDTEventTable Class

Overview:

This class is used to store ESDT events that are created during startup by the DsDeESDTDescriptor. In addition, when an ESDT notices that an event has occurred, this class is used to restore those events. This class is implemented as a Roguewave vector of DsSbEvents.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.17.2.7 DsGeESDTServiceProvider Class

Overview:

This abstract base class provides all of the truly public methods for ESDTs and ESDT Wrappers. All ESDTs and ESDTWrappers inherit from this class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DsGeESDTServiceProvider();`

BEGIN_PROLOG

NAME DsGeESDTServiceProvider

DESCRIPTION This is the default constructor for this class.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

`virtual ~DsGeESDTServiceProvider();`

default constructor BEGIN_PROLOG

NAME ~DsGeESDTServiceProvider

DESCRIPTION The destructor for the DsGeESDTServiceProvider.

RETURNS N/A

GLOBALS N/A

FILES N/A

END_PROLOG

Operations:

- Acquire

```
virtual EcUtStatus Acquire(const GlParameterList& ,
GlParameterList& failed, RWTPtrOrderedVector<<exp>>& ) = 0;
```

ParameterList for results

- ExecuteCommand

```
virtual EcUtStatus ExecuteCommand(const DsShSciCommandImp& ,
GlParameterList& ) = 0;
```

destructor

- GetmyUR

```
virtual GlUR GetmyUR() const = 0;
```

Output. Results of command

- Insert

```
virtual EcUtStatus Insert(const GlParameterList& ,
GlParameterList& ) = 0;
```

Get UR for this instance

4.3.17.2.8 DsGeESDTWrapper Class

Overview:

This class encapsulates the ESDT functionality and provides the public interface for access to all ESDT services. Upon construction, this class uses the ESDT configuration to determine which DLL should be loaded. Then the DLL is loaded and a new instance of the correct type is created. From then on, functions that are meant for the ESDT are passed through to it.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeESDTServiceProvider

Attributes:

`myDLL: DsGeESDTDynamicLibrary*`

the actual ESDT

`myESDT: DsGeESDT*`

Input. Name of the ESDT

Constructors and Destructor:

`DsGeESDTWrapper();`

`DsGeESDTWrapper(DsMdMetadata& metEntry, GlParameterList& results);`

command that has datatype information

`DsGeESDTWrapper(const DsGeESDTWrapper& );`

Output. ParameterList for results these are not implemented yet, but prevent inadvertant use

`DsGeESDTWrapper(const DsShSciCommandImp& cmd);`

`~DsGeESDTWrapper();`

results of ESDT creation BEGIN_PROLOG

NAME ~DsGeESDTWrapper

DESCRIPTION This is the destructor for this class. It calls a hidden member function to destroy any allocated memory.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- Acquire

`EcUtStatus Acquire(const GlParameterList& CmdParams, GlParameterList& failed, RWTPtrOrderedVector<<exp>>& results);`

Output. ParameterList for results BEGIN_PROLOG

NAME Acquire

DESCRIPTION This function is a pass-thru to the ESDT. It is used for the distribution of a granule

INPUTS results - parameterlist containing a list of files that make up this ESDT.

OUTPUTS None

RETURNS A EcUtStatus indicating success or failure of the acquire.

GLOBALS None

FILES None

END_PROLOG

- AttachDLL

```
void AttachDLL(const RWCString& datatype);
```

clears memory allocated for this ESDT BEGIN_PROLOG

NAME AttachDLL

DESCRIPTION This function uses the given string to determine the name of the DLL to attach to. This function also uses two environment variables: DSSCONFIG - Full path name of the Data Server Configuration File DSSDLLDIR - Directory that all DLLs reside in The DSSCONFIG is used to determine the name of the DLL file and the DSSDLLDIR is prepended to that name. Once all of this has been determined, the DLL is loaded and an actual ESDT is created.

INPUTS datatype - Name of the datatype we are about to create

OUTPUTS results - A parameter list to put select metadata attributes in. These are the attributes that are cached in the client space.

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- Clear

```
void Clear();
```

hidden default constructor BEGIN_PROLOG

NAME Clear

DESCRIPTION

This hidden member function is used to clear all memory allocated for this ESDT

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- ExecuteCommand

```
virtual EcUtStatus ExecuteCommand(const DsShSciCommandImp& cmd,
GlParameterList& plist);
```

Input. Instance on right hand side of < BEGIN_PROLOG

NAME ExecuteCommand

DESCRIPTION This member function is used to perform the given command and return the results in the given parameter list. The command is passed through to this wrappers ESDT. This is the member function that affords us the ability to evolve by adding or changing commands without having to change the interface.

INPUTS cmd - command to execute plist - parameter list to store results in

OUTPUTS plist - parameter list to store results in

RETURNS status indicating whether the command was successful or not

GLOBALS None

FILES None

END_PROLOG

- GetmyUR

```
GlUR GetmyUR() const;
```

Input/Output. Stream to send msgs to BEGIN_PROLOG

NAME GetmyUR

DESCRIPTION This function gets the UR of the underlying ESDT.

INPUTS None

OUTPUTS None

RETURNS UR of the underlying ESDT or 0

GLOBALS None

FILES None

END_PROLOG

- Insert

```
EcUtStatus Insert(const GlParameterList& args, GlParameterList&
results);
```

Get UR for the ESDT instance BEGIN_PROLOG

NAME Insert

DESCRIPTION This function is a pass-thru to the ESDT. It is used for the insertion of a new granule

INPUTS args - parameter list containing arguments that are used to perform the insertion results - the parameter list that contains the results of the insert

OUTPUTS None

RETURNS A EcUtStatus indicating success or failure of the call.

GLOBALS None

FILES None

END_PROLOG

- operator <

```
int operator <(const DsGeESDTWrapper& rhs);
```

Input.Instance on right hand side of == BEGIN_PROLOG

NAME operator <

DESCRIPTION This is the less than operator. It is needed so that you can use various RW Template collection classes. There really is no concept of one ESDTWrapper being less than another. Therefore this operator is implemented by comparing pointers.

INPUTS rhs - An ESDTWrapper on the right-hand side of the < to test against

OUTPUTS None

RETURNS int - 1 if ESDTWrapper is < rhs 0 if ESDTWrapper is >= rhs

GLOBALS None

FILES None

END_PROLOG

- operator ==

```
int operator ==(const DsGeESDTWrapper& rhs);
```

destructor the following are needed to support creation of a RW collection BEGIN_PROLOG

NAME operator==

DESCRIPTION This is the equality operator. It is needed so that you can use various RW Template collection classes. This is an identity comparison. Therefore two ESDTWrappers are equal if and only if their pointers are equal.

INPUTS rhs - An ESDTWrapper on the right-hand side of the == to test equality against

OUTPUTS None

RETURNS RWBoolean - Indicates whether the ESDTWrappers were equal or not

GLOBALS None

FILES None

END_PROLOG

- operator =

  ```
  DsGeESDTWrapper& operator =(const DsGeESDTWrapper& );
  ```

  Input. Instance to copy

- saveOn

  ```
  void saveOn(ostream& os) const;
  ```

  Output. Results parameter list BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION This member function is used for debug purposes. It allows you to print out the contents of this instance while in the debugger.

  INPUTS os - an output stream on which to write this ESDTWrappers contents

  OUTPUTS None

  RETURNS N/A

  GLOBALS None

  FILES None

  END_PROLOG

4.3.17.2.9 DsGeESDT Class

Overview:

The following classes are defined in this file: DsGeESDT ESDT base class

The following const variables are defined in this file: none

The following defines are defined in this file: DsGeESDTH ensures file isn't read twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following public functions are defined in this file: DsGeESDT() : DsGeESDTServiceProvider() default constructor ~DsGeESDT() default destructor

ExecuteCommand(const DsSrCommand &, GlParameterList &) parses the cmd param list

Fill(DsMdMetadata &, GlParameterList &) populate ESDT with metadata

SaveOn(ostream &os) send class attributes to ostream

GetmyUR() return UR of <this> ESDT

The following private functions are defined in this file: GetQueryableParameters(GlParameterList &) Return all queryable params for <this> ESDT

Validate(RWCString&, GlParameterList &) Validate metdata for <this> ESDT

Inspect(const DsSrCommand &,) Look at <this> ESDT metadata GlParameterList &

Update(GlParameterList &) Update metadata for <this> ESDT

Insert(const DsSrCommand &,); Insert an ESDT into SDSRV GlParameterList &

Acquire(const DsSrCommand &,); Acquire <this> ESDT RWTPtrOrderedVector<DsStFileParameters> & TriggreEvent(const RWCString&); Trigger a given event for this ESDT

This class is used to determine ESDT configuration information. It is derived from the DsCnConfiguration class which provides the basic opening, closing and getting token services. This class adds the services that are specific to ESDTs including parsing for specific well-defined tokens, determining the set of ESDTs currently available, and obtaining the DLL, descriptor, and event table handles.

This class performs runtime loading of ESDT libraries for a data type. It encapsulates the public functionality necessary for ESDT DLLs.

This file contains constants that are used by all classes in the Ge CSC.

This abstract base class provides all of the truly public methods for ESDTs and ESDT Wrappers. All ESDTs and ESDTWrappers inherit from this class.

This class encapsulates the ESDT functionality and provides the public interface for access to all ESDT services. Upon construction, this class uses the ESDT configuration to determine which DLL should be loaded. Then the DLL is loaded and a new instance of the correct type is created. From then on, functions that are meant for the ESDT are passed through to it.

Class definition

inherit from the public class - provides basic ESDT functionality Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeESDTServiceProvider

Attributes:

`myCatalogWrapper: DsSrGenCatalogWrapper`

Catalogue Object for <this> ESDT - one catalog connection / granule

`myDescriptor: DsDeDescriptor*`

Descriptor object for <this> ESDT - one reference per granule

myEventTable: DsDeEventTable*

myMetadata: DsMdMetadata*

Variables with class scope

Metadata object for <this> ESDT - one instance per granule.

myStatusFlag: EcUtStatus

Constructors and Destructor:

DsGeESDT(const DsShTypeID& type);

default constructor

DsGeESDT(const DsGeESDT&);

copy constructor defined here and not implemented to prevent anyone from using them.

virtual ~DsGeESDT();

virtual - makes sure the derived destructors get called BEGIN_PROLOG

NAME ~DsGeESDT

DESCRIPTION Default destructor for all ESDTs This function is declared virtual to ensure derived class destructors are called.

INPUTS none

OUTPUTS none

RETURNS none

END_PROLOG

default destructor

Operations:

- Acquire

virtual EcUtStatus Acquire(const GlParameterList& , GlParameterList& failed, RWTPtrOrderedVector<<exp>>& theresults);

Prepare granule for distribution BEGIN_PROLOG

NAME Acquire

DESCRIPTION Prepares <this> ESDT for distribution.

INPUTS GlParameterList & - not currently used.

OUTPUTS RWTPtrOrderedVector<DsStFileParameters> & - the results of the acquire (data location)

RETURNS EcUtStatus - Completion status

GLOBALS DsMdMetadata

FILES None

END_PROLOG

- ArchiveFiles

```
EcUtStatus ArchiveFiles(const GlParameterList& insertGroupList,
GlParameterList& ArchiveInputAndResults);
```

ESDT method for archiving files. Puts results into metadata BEGIN_PROLOG

NAME ArchiveFiles

DESCRIPTION Provides generic functionality for derived ESDTs to archive 'their' files. If file archiving is successful relevant attributes are added to metadata.

INPUTS GlParameterList& - list of files to archive

OUTPUTS ArchiveInputAndResults& - list of those files archived. Includes filename, checksum, filesize, status (Success, Failure), Description (Reason for failure)

RETURNS EcUtStatus - Completion status

GLOBALS none

FILES myFileType - accesses those files specified within insertGroupList

NOTES Failure to archive any one file will cause failure of the whole method. This method assumes all files within insertGroupList are all constituents parts of ONE granule. The method should not be used to archive files representing different data granules.

END_PROLOG

- BaseType

```
virtual enum DsGeOID::_basetypetag BaseType() const = 0;
```

Get ESDT basetype - this function MUST be defined by the base type ESDT's - ScienceData, QA, Browse etc.

- Clear

```
virtual void Clear();
```

Clear out the private member variables of this ESDT BEGIN_PROLOG

NAME Clear

DESCRIPTION Clear out all of the member variables of an ESDT instance.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- ConstructedOK

```
EcUtStatus ConstructedOK() const;
```

Check that ESDT constructed successfully BEGIN_PROLOG

NAME ConstructedOK

DESCRIPTION This member function indicates whether this ESDT was successfully constructed or not.

RETURNS EcUtStatus indicating success or failure

END_PROLOG

- ExecuteCommand

```
virtual EcUtStatus ExecuteCommand(const DsShSciCommandImp& cmd,
GlParameterList& cmdresults);
```

I: An ESDT Execute an ESDT specific service BEGIN_PROLOG

NAME ExecuteCommand

DESCRIPTION Generic execute command for all ESDT's. This is provided for those ESDT's that have no more than the basic core set of ESDT services (Acquire, Insert and Inspect).

INPUTS const DsSrCommand& - reference to command object, contains name of ESDT service to execute and parameterlist of associated command parameters.

OUTPUTS GlParameterList& - reference to parameter list into which results of command execution are placed (contents dependent on type of command) RETURNS EcUtStatus - Completion status of command

GLOBALS none

FILES none

NOTES None

END_PROLOG

- Fill

```
virtual EcUtStatus Fill(DsMdMetadata& ResultsMetadata,
GlParameterList& cmdresults);
```

O: List of results Associate metadata with ESDT instance - i.e 'fill' ESDT with metadata The return parameters from this function will be revisited at phase 2 BEGIN_PROLOG

NAME Fill

DESCRIPTION "Fills" <this> ESDT with related metadata. Passes a metadata object containing metadata for <this> ESDT, the method returns a subset of the metadata to the calling class (usually the ESDT wrapper).

INPUTS DsMdMetadata& - metadata object to associate with <this> ESDT

OUTPUTS GlParameterList& - reference to parameter list into which subset of metadata is returned to calling class

RETURNS EcUtStatus - Completion status of fill

GLOBALS myMetadata - pointer to <this> ESDT metadata object.

FILES none

NOTES none

END_PROLOG

- GetmyUR

```
virtual GlUR GetmyUR() const;
```

O: Output stream Retrieve ESDT UR BEGIN_PROLOG

NAME GetmyUR

DESCRIPTION Get the UR of <this> ESDT.

INPUTS none

RETURNS GlUR - the UR object for this ESDT

GLOBALS myMetadata

FILES None

END_PROLOG

- GetQueryableParameters

```
EcUtStatus GetQueryableParameters(GlParameterList& ) const;
```

Extract queryable parameters for <this> ESDT

- GetSize

```
virtual long GetSize() const;
```

Get size of ESDT - constitutes sum of the size of all files BEGIN_PROLOG

NAME Size

DESCRIPTION Get the size of this instance.

RETURNS Long containing this instances size

GLOBALS None

FILES None

END_PROLOG

- GetUniqueFilename

`EcUtStatus GetUniqueFilename(const GlStringP& , GlStringP& theUniqueFileName) const;`

Generate a unique filename BEGIN_PROLOG

NAME GetUniqueFilename

DESCRIPTION Generates a unique filename for the archive.

INPUTS GlStringP* theOrigFileName

OUTPUTS GlStringP* theUniqueFileName -- must not include a path!

RETURNS EcUtStatus - Completion status

GLOBALS none

NOTES Not for phase 2 review. This function provides a 'hook' for future development following project decisions regarding file naming conventions

END_PROLOG

- Insert

`virtual EcUtStatus Insert(const GlParameterList& CommandParams, GlParameterList& insertResults);`

O: List of results Add a new granule to the dataserver The list of parameters depends on the datatype BEGIN_PROLOG

NAME Insert

DESCRIPTION Insert <this> ESDT into the SDSRV. Provides that insert functionality common to all ESDTs. Validates the command parameter list using <this> ESDTs Descriptor and adds new UR to metadata.

INPUTS GlParameterList& - parameter list containing insert filenames. The parameters are product specific,including names of science file names, product metadata et al.

OUTPUTS GlParameterList& - reference to parameter list into which the new UR(s) of inserted granule(s) are placed.

RETURNS EcUtStatus - Completion status

GLOBALS myMetadata - pointer to new ESDT metadata object.

FILES none

NOTES NonScience ESDT metadata is sent to the DB immediately. Science granules must 'wait' until all associated ESDTs within the insert command (Browse, QA etc) have been successfully archived, and 'their' ID's added to the science data granule metadata. Only then may metadata DB insertion procede.

END_PROLOG

- Inspect

`virtual EcUtStatus Inspect(const GlParameterList& CommandParams, GlParameterList& cmdresults) const;`

Inspect BEGIN_PROLOG

NAME Inspect

DESCRIPTION Facilitates the inspection of <this> ESDTs metadata attributes. the user of this method specifies which attributes are to be obtained

INPUTS const GlParameterList & - list of metadata attributes for which values will be obtained

OUTPUTS GlParameterList & - the results of the inquiry. These results are newly created parameters that the caller is responsible for deleting.

RETURNS EcUtStatus - Completion status

GLOBALS DsMdMetadata

FILES None

NOTES The new UR generated is added to <this> ESDT metadata. The calling routine is responsible for ensuring memory allocation of the cmdresults.

END_PROLOG

- LoadMetadataFromFile

```
EcUtStatus LoadMetadataFromFile(const GlParameterList&
CommandParams, GlParameterList& insertResults);
```

O: Unique filename Open metadatafile, load to DsMdMetadata object BEGIN_PROLOG

NAME LoadMetadataFromFile

DESCRIPTION Finds and opens specified metadata file within param list & calls Descriptor parsing

INPUTS GlParameterList& CommandParams

OUTPUTS none

RETURNS EcUtStatus - Completion status

GLOBALS myMetadata

FILES None

END_PROLOG

- operator ==

```
int operator ==(const DsGeESDT& rhs) const;
```

Overloaded equality operator BEGIN_PROLOG

NAME operator ==

DESCRIPTION Overload the equality operator. Compares memory addresses.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- operator =

```
DsGeESDT& operator =(const DsGeESDT& );
```

I: An ESDT assignment operator

- saveOn

```
void saveOn(ostream& os) const;
```

O: Results of fill Save internals of ESDT on output stream BEGIN_PROLOG

NAME saveOn

DESCRIPTION Save metadata of <this> ESDT to ostream.

INPUTS ostream

OUTPUTS ostream - stream to output results on

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- SetNewUR

```
EcUtStatus SetNewUR();
```

O: Results of archive per file Get a new UR for <this> ESDT & put into <this> ESDT metadata. BEGIN_PROLOG

NAME SetNewUR

DESCRIPTION Used to generate a new unique identifier for <this> ESDT. Primarily used during an insertion of a new granule, or as a result of a service deriving a new granule from <this> granule (e.g subsetting).

INPUTS GlParameterList& CommandParams - contains basetype & subtype of <this> ESDT required for generation of GeOID object

OUTPUTS none

RETURNS EcUtStatus - Completion status

GLOBALS DsMdMetadata

FILES None

NOTES The new UR generated is added to <this> ESDT metadata. For phase 2 we simply create a GeOID object containing basetype (e.g Science), subtype (e.g AVHRR), dbid (unique id generated by database). During phase 3 this method will inaddition use appropriate classes to derive a granule UR, for eventual return to the user.

END_PROLOG

- SubType

  ```
  virtual RWCString SubType() const = 0;
  ```

  Get ESDT type - this function MUST be defined by specialised ESDT's Avhrr, CERES02 etc

- TriggerEvent

  ```
  void TriggerEvent(const RWCString& EventName);
  ```

  O: List of results Trigger an event BEGIN_PROLOG

  NAME TriggerEvent

  DESCRIPTION Trigger an event for this ESDT.

  INPUTS RWCString EventName A predefined constant from DsGeServiceNames.h (e.g. DsCGeInsert)

  OUTPUTS Trigger mechanism is not implememted yet. The ouput will be whatever output occurs from a trigger by the subsriptin server.

  RETURNS None

  GLOBALS myEventTable

  FILES

  NOTES To ensure consistency, a constant from DsCGeServiceNames is recommended for the argument

  END_PROLOG

- Update

  ```
  EcUtStatus Update(GlParameterList& glParmIn);
  ```

  O: List of results Update with new Metadata values BEGIN_PROLOG

  NAME Update

  DESCRIPTION Update this instance with new metadata

  RETURNS Status indicating success or failure

  GLOBALS None

  FILES None

  END_PROLOG

- Validate

  ```
  EcUtStatus Validate(RWCString& , GlParameterList& ) const;
  ```

  I: list of querable parameters Validate the Metadata for <this> ESDT

4.3.17.2.10 DsGeOID Class

Overview:

This class provides a mechanism for uniquely identifying a data granule within a DataServer.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myBaseType: enum DsGeOID::_basetypetag`

assignment operator [not-implemented]

`myID: DsMdID*`

this IDs sub type

`mySubType: RWCString`

this IDs base type

`ourValidsSet: int`

this IDs database identifier static flag indicating whether valids have been initialized

`ourValids: RWTValHashDictionary<class RWCString, enum DsGeOID::_basetypetag>*`

static collection containing the valid base types

Constructors and Destructor:

`DsGeOID(const DsGeOID& rhs);`

Input/Output. Stream to send msgs to

`DsGeOID(const enum DsGeOID::_basetypetag baseType, RWCString* subType);`

Input. make from string

`DsGeOID(const enum DsGeOID::_basetypetag baseType, RWCString subType, RWCString& theID);`

Input. this id's subtype (optional)

`DsGeOID(const RWCString& inStr);`

```
virtual  ~DsGeOID();
```

copy constructor BEGIN_PROLOG

NAME ~DsGeOID

DESCRIPTION This is the destructor for this class.

INPUTS N/A

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- BaseSetValids

```
static void BaseSetValids();
```

BEGIN_PROLOG

NAME BaseSetValids

DESCRIPTION This static member function is used to ensure that a vector containing the valid values for base type is created.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- GetBaseType

```
RWCString GetBaseType() const;
```

Return this as a string BEGIN_PROLOG

NAME GetBaseType

DESCRIPTION This member function is used to obtain the base type as a human-readable string.

INPUTS None

OUTPUTS None

RETURNS RWCString with this DsGeOIDs base type as a string.

GLOBALS None

FILES None

END_PROLOG

- GetDBID

```
const DsMdID* GetDBID() const;
```

Return this id's subtype BEGIN_PROLOG

NAME GetDBID

DESCRIPTION This member function gets the database identifier object of this DsGeOID.

INPUTS None

OUTPUTS None

RETURNS A constant pointer to a DsMdID object

GLOBALS None

FILES None

END_PROLOG

- GetSubType

```
RWCString GetSubType() const;
```

Return this id's base type BEGIN_PROLOG

NAME GetSubType

DESCRIPTION This member function is used to obtain a human-readable string containing this DsGeOIDs subtype.

INPUTS None

OUTPUTS None

RETURNS RWCString containing the subtype

GLOBALS None

FILES None

END_PROLOG

- operator ==

```
int operator ==(const DsGeOID& rhs);
```

Input. this id's database identifier BEGIN_PROLOG

NAME operator==

DESCRIPTION This member function checks to see if two DsGeOIDs are equal

INPUTS rhs - a DsGeOID instance that was on the right-hand side of the ==

OUTPUTS N/A

RETURNS RWBoolean indicating TRUE if they are equal or FALSE otherwise.

GLOBALS None

FILES None

END_PROLOG

- operator =

```
DsGeOID& operator =(const DsGeOID& );
```

Create set of valid base types

- saveOn

```
void saveOn(ostream& os) const;
```

Set this id's subtype BEGIN_PROLOG

NAME saveOn

DESCRIPTION This member function is used for debug purposes. It allows you to print out the contents of this instance while in the debugger.

INPUTS os - an output stream on which to write this DsGeOID's contents

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- SetSubType

```
void SetSubType(RWCString st);
```

Return object that has the database identifier BEGIN_PROLOG

NAME SetSubType

DESCRIPTION This method is used to set the subtype portion of the DsGeOID

INPUTS st - an RWCString containing the new value for subtype

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

- Textify

```
RWCString Textify() const;
```

Input. Right-hand side of double equals BEGIN_PROLOG

NAME Textify

DESCRIPTION This member function is used to print out the DsGeOID in a human-readable form.

INPUTS None

OUTPUTS N/A

RETURNS RWCString containing this DsGeOID as a human-readable entity.

GLOBALS None

FILES None

END_PROLOG

4.3.17.2.11 DsGeScienceData Class

Overview:

The following classes are defined in this file: DsGeScienceData

The following const variables are defined in this file: none

The following defines are defined in this file: DsGeScienceDataH ensures file isn't read twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following public functions are defined in this file: DsGeScienceData(); ctor ~DsGeScienceData(); dtor virtual EcUtStatus ExecuteCommand(const DsSrCommand &, GlParameterList &) parses the cmd param list

The following protected functions are defined in this file: virtual EcUtStatus Insert(const GlParameterList&, GlParameterList&); virtual RWCString SubType() const; Get ESDT type virtual EcUtStatus AcquireBrowse(Prepare browse for const GlParameterList&, distribution GlParameterList &); Provide the browse data associated with this science data virtual EcUtStatus Browse(GlParameterList&); Get ESDT base type - defined at this level of specialization virtual DsEGeBaseType BaseType() const;

inherit from the public ESDT class, which provides basic ESDT functionality

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeESDT

Attributes:

Constructors and Destructor:

```
DsGeScienceData(const DsShTypeID& type);
```

```
virtual  ~DsGeScienceData();
```

BEGIN_PROLOG

NAME ~DsGeScienceData

DESCRIPTION This is the default destructor for the DsGeScienceData datatype.

INPUTS None

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- AcquireBrowse

```
virtual EcUtStatus AcquireBrowse(const GlParameterList& ,
GlParameterList& theresults);
```

Prepare browse for distribution BEGIN_PROLOG

NAME AcquireBrowse

DESCRIPTION Prepares the browse associated with <this> ESDT for distribution.

INPUTS none

OUTPUTS GlParameterList & - the results of the acquire (data location)

RETURNS EcUtStatus - Completion status

GLOBALS DsMdMetadata

FILES NONE

NOTES NONE

END_PROLOG

- BaseType

```
virtual enum DsGeOID::_basetypetag BaseType() const;
```

Get ESDT base type - defined at this level of specialization BEGIN_PROLOG

NAME BaseType

DESCRIPTION Returns subtype of <this> ESDT.

INPUTS None

OUTPUTS None

RETURNS DsEGeBaseType - enum defined by GeOID class

GLOBALS None

FILES None

END_PROLOG

- Browse

```
virtual EcUtStatus Browse(GlParameterList& BrowseResults);
```

O: List of results Provide the browse data associated with this science data BEGIN_PROLOG

NAME Browse

DESCRIPTION Return to the user a GlBinaryP parameter containing the Browse ESDT associated with <this> ESDT

INPUTS none

OUTPUTS GlParameterList& - reference to parameter list into which the binary browse parameter is placed RETURNS EcUtStatus - Completion status of command

GLOBALS None

FILES None

NOTES None

END_PROLOG

- ExecuteCommand

```
virtual EcUtStatus ExecuteCommand(const DsShSciCommandImp& cmd,
GlParameterList& cmdresults);
```

BEGIN_PROLOG

NAME ExecuteCommand

DESCRIPTION Execute command for science data. Looks for Browse service request.

INPUTS const DsSrCommand& - reference to command object, contains name of ESDT service to execute and parameterlist of associated command parameters.

OUTPUTS GlParameterList& - reference to parameter list into which results of command execution are placed (contents dependent on type of command) RETURNS EcUtStatus - Completion status of command

GLOBALS none

FILES none

NOTES None

END_PROLOG

- Insert

```
virtual EcUtStatus Insert(const GlParameterList& commandParams,
GlParameterList& insertResults);
```

BEGIN_PROLOG

NAME Insert

DESCRIPTION Generic execute command for science based ESDT's. This is provided for those science data products that do not require a data specific insert service. The service is responsible for building those science data associations - DsNsQAStatistics, DsNsProductionHistory, DsGeBrowseProduct.

INPUTS const DsSrCommand& - reference to command object, contains name of ESDT service to execute and parameterlist of associated command parameters.

OUTPUTS GlParameterList& - reference to parameter list into which results of command execution are placed (contents dependent on type of command) RETURNS EcUtStatus - Completion status of command

GLOBALS None

FILES None

NOTES None

END_PROLOG

- SubType

```
virtual RWCString SubType() const = 0;
```

Get ESDT type - this function must be defined by specialized ESDT's

4.3.17.2.12 DsGeSummaryProduct Class

Overview:

A summary product is an ECS data product that represents a summary of th einformatio contained in another ECS data product. For example, a monthly product is an ECS data product AND it may also be the summary product for a daily product.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

```
myGranuleList: GlUR
```

A list of URs that are summarized by this summary product.

Constructors and Destructor:

Operations:

- AddGranule

  ```
  AddGranule(GlUR & data);
  ```

  This service adds the given granule to the list of granules summarized by this instance of a summary product.

- ListGranules

  ```
  GlUR 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.

- RemoveGranule

  ```
  RWBoolean RemoveGranule();
  ```

  This service is used to remove granules from the list of granules summarized by this instance of a summary product.

4.3.17.2.13 DsGeSuspendedSession Class

Overview:

Distributed Object: No

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsGeSuspendedSession();

~DsGeSuspendedSession();
```

Operations:

- Insert

```
Insert();
```

- SaveRequests

```
SaveRequests();
```

- SaveWorkingCollection

```
SaveWorkingCollection();
```

4.3.17.2.14 DsGeTypeID Class

Overview:

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.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myCode: unsigned long`

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.

`myName: RWCString`

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.

`myVersionB: RWCString`

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

Constructors and Destructor:

```
DsGeTypeID(RWCString &theCode, =NULL RWCString &theName,
RWCString theVersion=NULL);
```

This constructor creates a TypeID instance from the given code. This code uniquely identifies a TypeID. 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.

```
~DsGeTypeID();
```

The destructor for this type has no implementation at this time.

Operations:

- GetTypeCode

```
unsigned long & GetTypeCode();
```

Returns the code for this TypeID.

- GetTypeName

```
RWCString GetTypeName();
```

Returns the myName attribute which is the type name of this instance.

- GetTypeVersionB

```
RWCString GetTypeVersionB();
```

Returns the string contained in the myVersion attribute. This is the version of this TypeID.

- SetTypeCode

```
SetTypeCode(unsigned long);
```

This is a private member function used to set the value of the myCode attribute.

- SetTypeName

```
SetTypeName(RWCString &);
```

This is a private member function used to set the value of the myName attribute.

- SetTypeVersionB

```
SetTypeVersionB(RWCString &);
```

This is a private member function used to set the value of the myVersion attribute.

4.3.18 gl Class Category

4.3.18.1 Overview

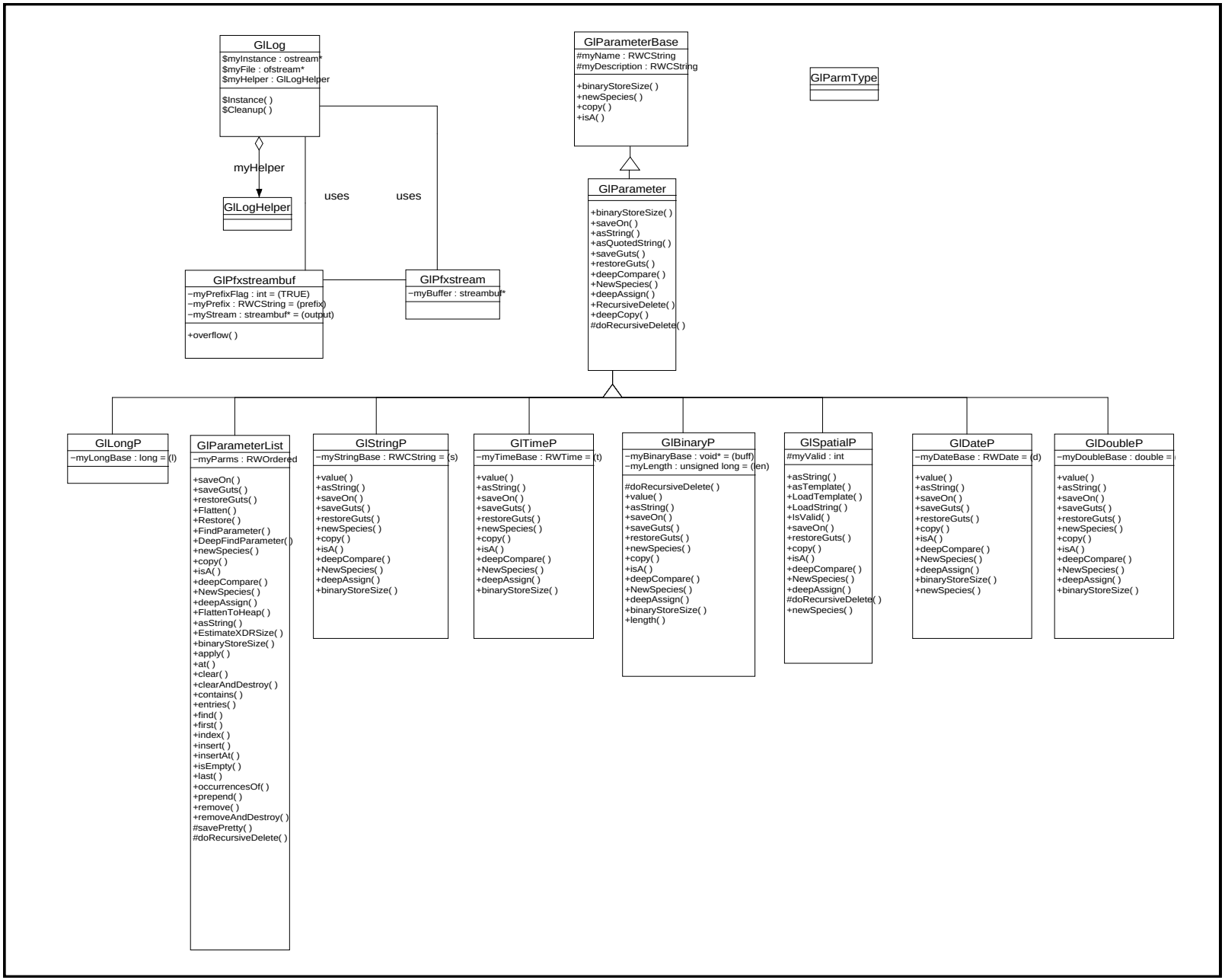

GILog
$myInstance : ostream*
$myFile : ofstream*
$myHelper : GILogHelper
$Instance( )
$Cleanup( )
myHelper
GILogHelper
uses
uses
GIPfxstreambuf
−myPrefixFlag : int = (TRUE)
−myPrefix : RWCString = (prefix)
−myStream : streambuf* = (output)
+overflow( )
GIPfxstream
−myBuffer : streambuf*
GIParameterBase
#myName : RWCString
#myDescription : RWCString
+binaryStoreSize( )
+newSpecies( )
+copy( )
+isA( )
GIParmType
GIParameter
+binaryStoreSize( )
+saveOn( )
+asString( )
+asQuotedString( )
+saveGuts( )
+restoreGuts( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
+RecursiveDelete( )
+deepCopy( )
#doRecursiveDelete( )
GILongP
−myLongBase : long = (l)
GIParameterList
−myParms : RWOrdered
+saveOn( )
+saveGuts( )
+restoreGuts( )
+Flatten( )
+Restore( )
+FindParameter( )
+DeepFindParameter( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
+FlattenToHeap( )
+asString( )
+EstimateXDRSize( )
+binaryStoreSize( )
+apply( )
+at( )
+clear( )
+clearAndDestroy( )
+contains( )
+entries( )
+find( )
+first( )
+index( )
+insert( )
+insertAt( )
+isEmpty( )
+last( )
+occurrencesOf( )
+prepend( )
+remove( )
+removeAndDestroy( )
#savePretty( )
#doRecursiveDelete( )
GIStringP
−myStringBase : RWCString = (s)
+value( )
+asString( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
+binaryStoreSize( )
GITimeP
−myTimeBase : RWTime = (t)
+value( )
+asString( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
+binaryStoreSize( )
GIBinaryP
−myBinaryBase : void* = (buff)
−myLength : unsigned long = (len)
#doRecursiveDelete( )
+value( )
+asString( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
+binaryStoreSize( )
+length( )
GISpatialP
#myValid : int
+asString( )
+asTemplate( )
+LoadTemplate( )
+LoadString( )
+IsValid( )
+saveOn( )
+restoreGuts( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
#doRecursiveDelete( )
+newSpecies( )
GIDateP
−myDateBase : RWDate = (d)
+value( )
+asString( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
+binaryStoreSize( )
+newSpecies( )
GIDoubleP
−myDoubleBase : double =
+value( )
+asString( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
+binaryStoreSize( )

Figure 28: IO

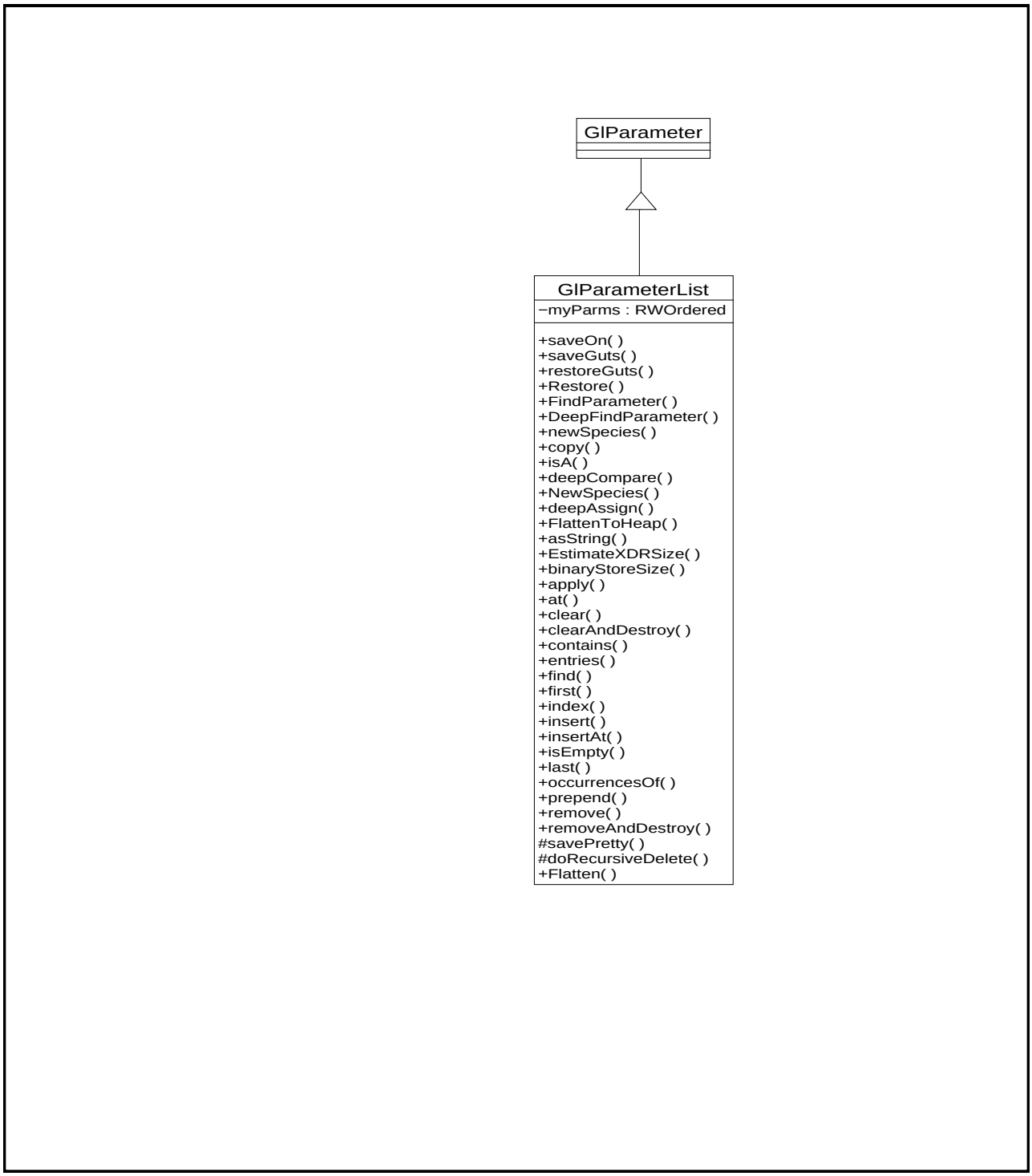


Figure 29: GlI

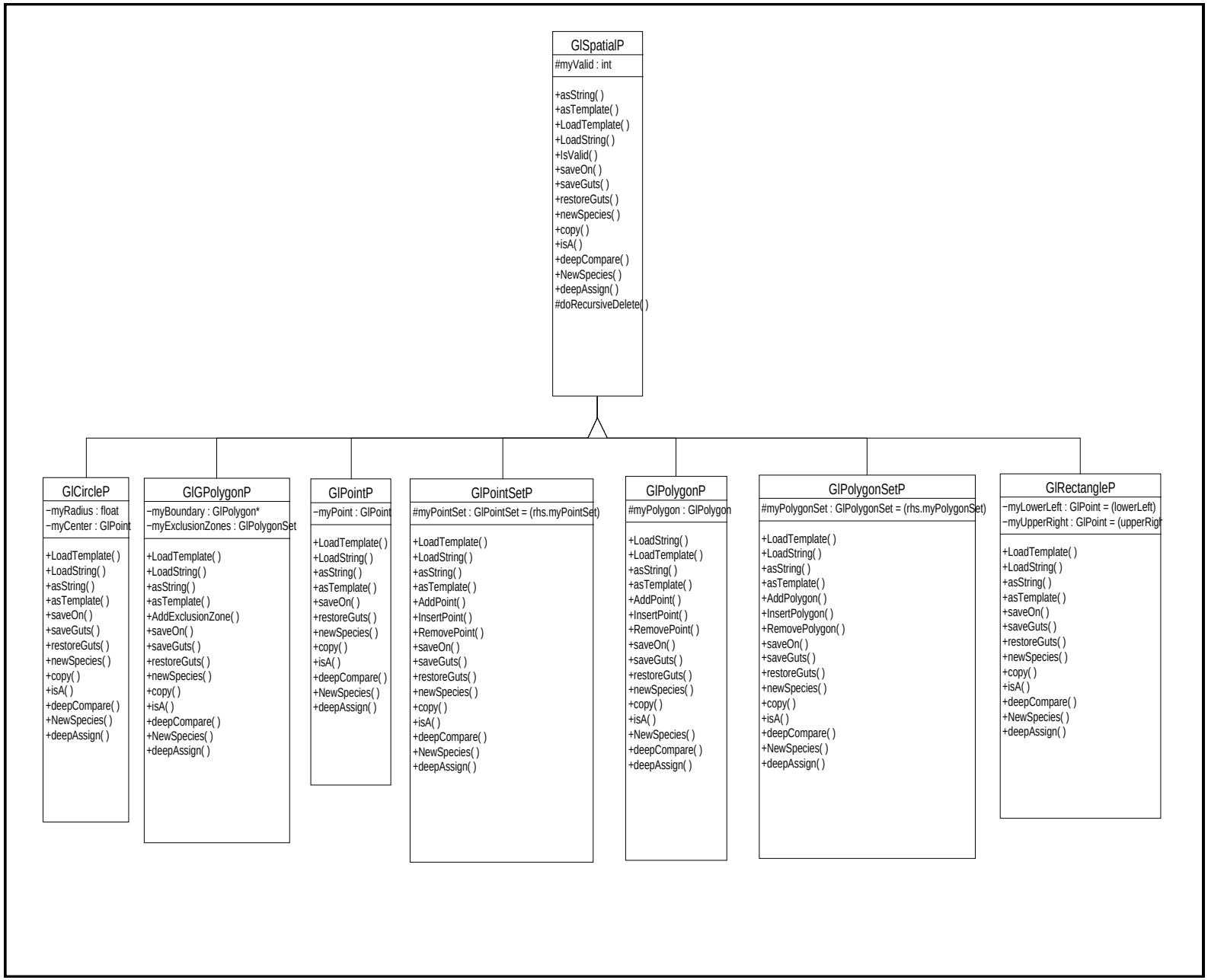

GlSpatialP
#myValid : int
+asString( )
+asTemplate( )
+LoadTemplate( )
+LoadString( )
+IsValid( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
#doRecursiveDelete( )
GlCircleP
−myRadius : float
−myCenter : GlPoint
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlGPolygonP
−myBoundary : GlPolygon*
−myExclusionZones : GlPolygonSet
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+AddExclusionZone( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlPointP
−myPoint : GlPoint
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+saveOn( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlPointSetP
#myPointSet : GlPointSet = (rhs.myPointSet)
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+AddPoint( )
+InsertPoint( )
+RemovePoint( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlPolygonP
#myPolygon : GlPolygon
+LoadString( )
+LoadTemplate( )
+asString( )
+asTemplate( )
+AddPoint( )
+InsertPoint( )
+RemovePoint( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+NewSpecies( )
+deepCompare( )
+deepAssign( )
GlPolygonSetP
#myPolygonSet : GlPolygonSet = (rhs.myPolygonSet)
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+AddPolygon( )
+InsertPolygon( )
+RemovePolygon( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlRectangleP
−myLowerLeft : GlPoint = (lowerLeft)
−myUpperRight : GlPoint = (upperRigh
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )

Figure 30:12

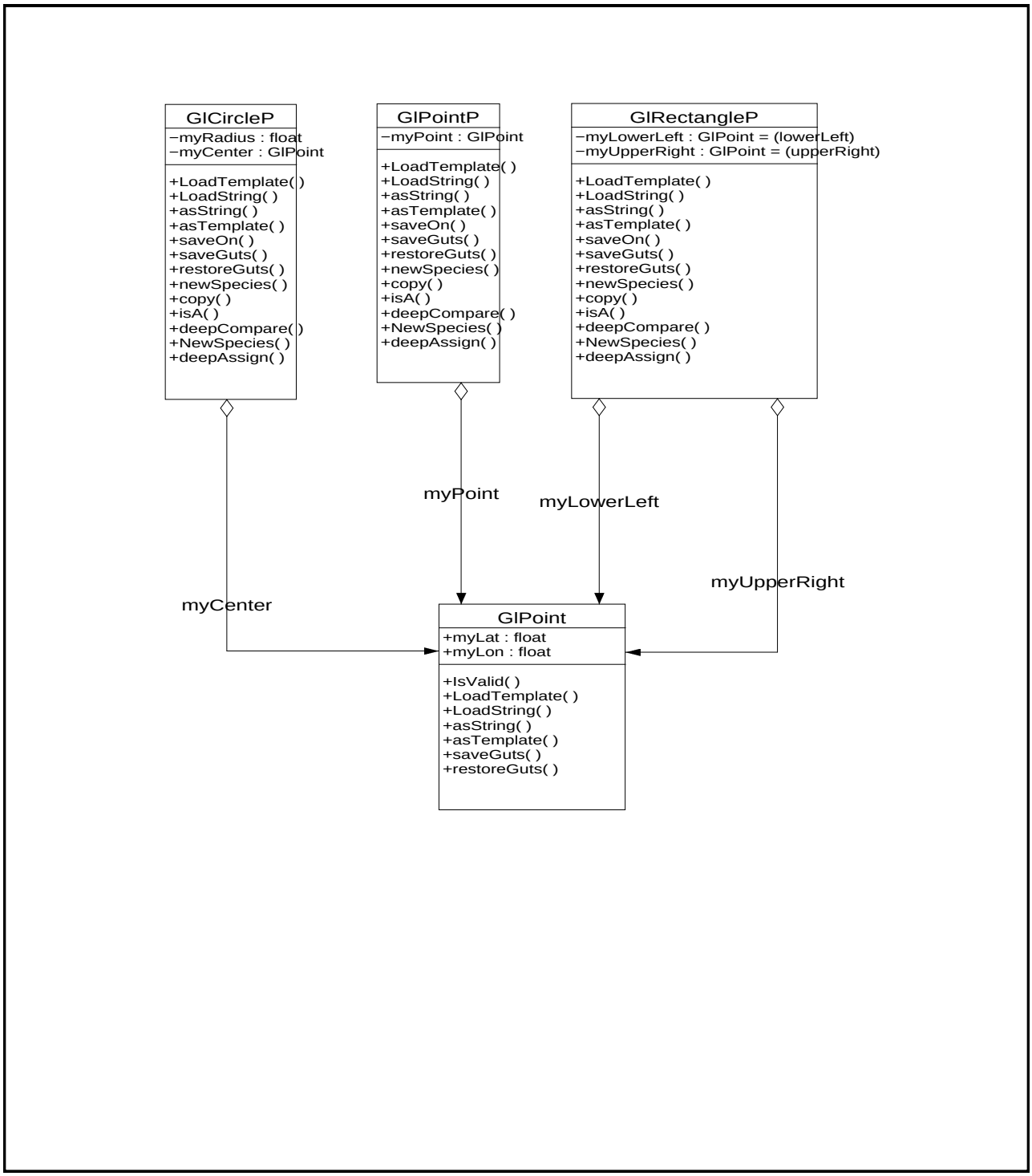

GlCircleP
−myRadius : float
−myCenter : GlPoint
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlPointP
−myPoint : GlPoint
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlRectangleP
−myLowerLeft : GlPoint = (lowerLeft)
−myUpperRight : GlPoint = (upperRight)
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
myPoint
myLowerLeft
myUpperRight
myCenter
GlPoint
+myLat : float
+myLon : float
+IsValid( )
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+saveGuts( )
+restoreGuts( )

Figure 31.13

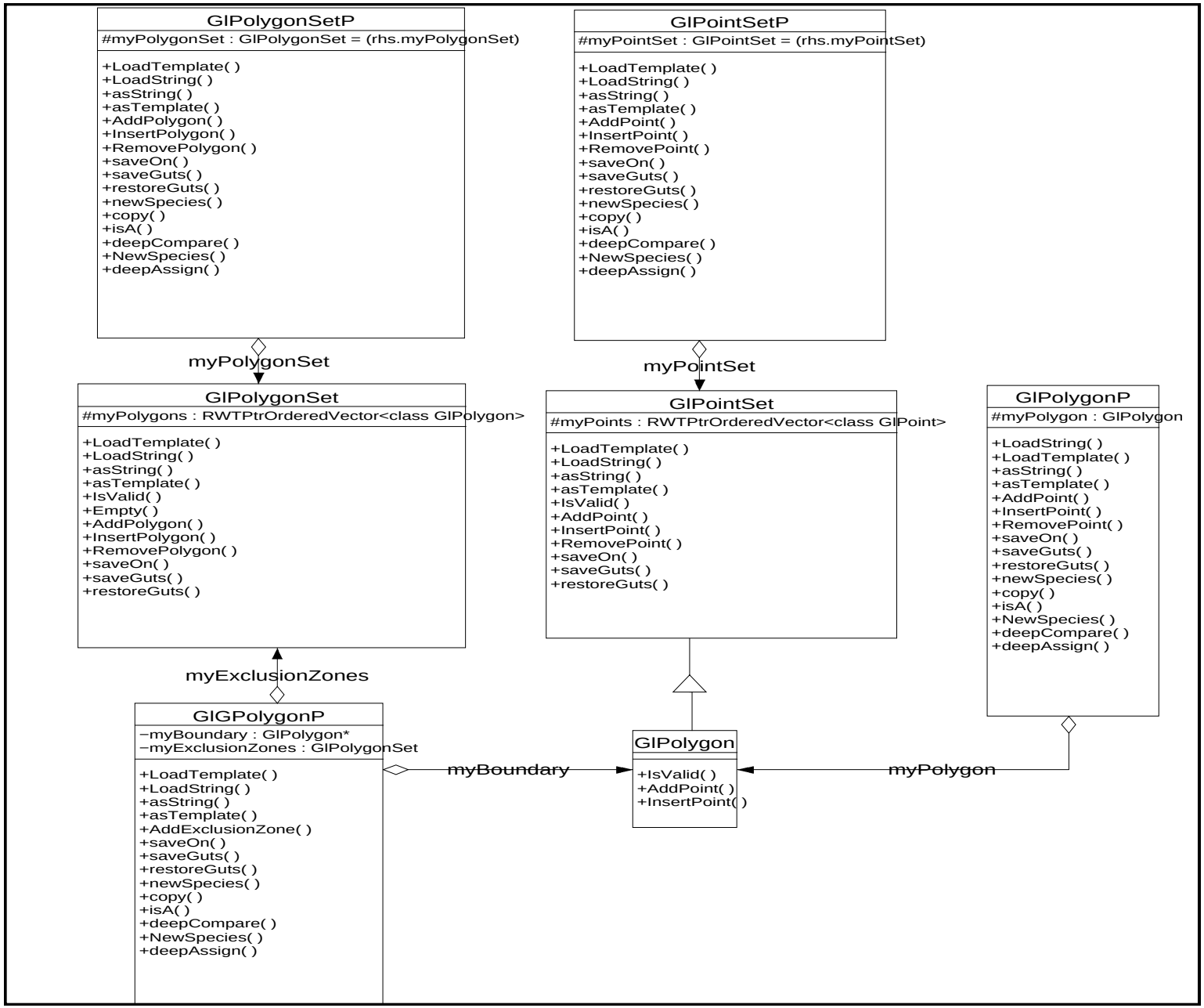

GlPolygonSetP
#myPolygonSet : GlPolygonSet = (rhs.myPolygonSet)
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+AddPolygon( )
+InsertPolygon( )
+RemovePolygon( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
GlPointSetP
#myPointSet : GlPointSet = (rhs.myPointSet)
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+AddPoint( )
+InsertPoint( )
+RemovePoint( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
myPolygonSet
myPointSet
GlPolygonSet
#myPolygons : RWTPtrOrderedVector<class GlPolygon>
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+IsValid( )
+Empty( )
+AddPolygon( )
+InsertPolygon( )
+RemovePolygon( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
GlPointSet
#myPoints : RWTPtrOrderedVector<class GlPoint>
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+IsValid( )
+AddPoint( )
+InsertPoint( )
+RemovePoint( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
GlPolygonP
#myPolygon : GlPolygon
+LoadString( )
+LoadTemplate( )
+asString( )
+asTemplate( )
+AddPoint( )
+InsertPoint( )
+RemovePoint( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+NewSpecies( )
+deepCompare( )
+deepAssign( )
myExclusionZones
GlGPolygonP
−myBoundary : GlPolygon*
−myExclusionZones : GlPolygonSet
+LoadTemplate( )
+LoadString( )
+asString( )
+asTemplate( )
+AddExclusionZone( )
+saveOn( )
+saveGuts( )
+restoreGuts( )
+newSpecies( )
+copy( )
+isA( )
+deepCompare( )
+NewSpecies( )
+deepAssign( )
myBoundary
GlPolygon
+IsValid( )
+AddPoint( )
+InsertPoint( )
myPolygon

Figure 32: 1.4

4.3.18.2 gl Classes

4.3.18.3 GlBinaryP Class

Overview:

The following classes are defined in this file: GlBinaryP specialization of GlParameter for binary data

The following const variables are defined in this file: none

The following defines are defined in this file: GlBinaryPH make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlBinaryP(); default constructor needed for RWCollectable GlBinaryP(const GlBinaryP&); copy constructor GlBinaryP(void* buff, normal constructor parms = unsigned long len, value, length, and name const char* name = 0); virtual ~GlBinaryP(); destructor GlBinaryP& operator=(assignment operator const GlBinaryP&); copy from a GlBinaryP void* value() const; retrieve parameter's value virtual RWCString asString() const; convert to string void saveOn(human-readable output ostream &) const; stream on which to write void saveGuts(XDR, etc. output RWvostream&) const; stream on which to write void saveGuts(XDR, etc. output RWFile&) const; file to which to write void restoreGuts(XDR, etc. input RWvistream&); stream from which to read void restoreGuts(XDR, etc. input RWFile&); file from which to read RWBoolean deepCompare(compare base value const GlParameter&) const; object with which to compare GlParameter* NewSpecies()const; create new GlBinaryP pointed to as a GlParameterP void deepAssign(copy including base value const GlParameter&); generic parameter to copy RWspace binaryStoreSize() const; calculate store size including base value size void doRecursiveDelete(clean up heap storage RWTValOrderedVector<GlParameter*>&); list to work over GlBinaryP& Set(void*, unsigned long); to change value unsigned long length() const; return length of value

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

`myBinaryBase: void*`

`myLength: unsigned long`

Constructors and Destructor:

```
GlBinaryP(void* buff, unsigned long len, const char* name);
```

copy constructor

```
GlBinaryP(const GlBinaryP& rhs);
```

default constructor needed for RWCollectable

```
GlBinaryP();
```

```
virtual  ~GlBinaryP();
```

Operations:

- asString

```
virtual RWCString asString() const;
```

retrieve parameter's value BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation provides the capability to be compatible with other GlParameter types. This operation just returns a string indicating that the value for this object is BINARY.

RETURNS RWCString, the output from the value() operation.

GLOBALS none

FILES none

END_PROLOG

- binaryStoreSize

```
unsigned long binaryStoreSize() const;
```

generic parameter to copy BEGIN_PROLOG

NAME binaryStoreSize() const

DESCRIPTION

This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

create new GlBinaryP pointed to as a GlParameterP BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, IFF the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

BEGIN_PROLOG

NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

DESCRIPTION

This operation performs a value comparison between the base values of the two objects.

RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

- doRecursiveDelete

```
void doRecursiveDelete(RWTValOrderedVector<class GlParameter*>&
);
```

BEGIN_PROLOG

NAME doRecursiveDelete(RWTValOrderedVector<GlParameter*>&)

DESCRIPTION

This operation cleans up the heap storage. It is required via a (protected) virtual function in base class GlParameter. This operation deallocates any heap storage pointed to by this object. This

object's implementation of this required function does nothing more than that; for the full story, see GlParameter's function RecursiveDelete().

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- length

```
unsigned long length() const;
```

to change value BEGIN_PROLOG

NAME length() const

DESCRIPTION

This operation returns the length of the base value of this object

RETURNS unsigned long, the length of the base value of this object

GLOBALS none

FILES none

END_PROLOG

- NewSpecies

```
GlParameter* NewSpecies() const;
```

object with which to compare BEGIN_PROLOG

NAME NewSpecies() const

DESCRIPTION

This operation creates a new instance of a GlBinaryP, but returns the pointer to the new object as a (generic) GlParameter*.

RETURNS GlParameter*, the pointer to the newly allocated GlBinaryP

GLOBALS none

FILES none

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- operator =

  `GlBinaryP& operator =(const GlBinaryP& rhs);`

  destructor BEGIN_PROLOG

  NAME operator=(const GlBinaryP& rhs) // right hand side

  DESCRIPTION

  This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match. (NOTE: it does a SHALLOW copy)

  RETURNS GlDateP&, reference to self

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWFile& file);`

  stream from which to read BEGIN_PROLOG

  NAME restoreGuts(RWFile& file) // file from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave file in RW-standard file format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWvistream& vis);`

  file to which to write BEGIN_PROLOG

  NAME restoreGuts(RWvistream& vis) // stream from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& file) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

convert to string BEGIN_PROLOG

NAME saveOn(ostream &os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a normal stream in a human-readable format

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- Set

```
GlBinaryP& Set(void* , unsigned long );
```

calculate store size including base value size BEGIN_PROLOG

NAME Set(void*, unsigned long)

DESCRIPTION

NOTE: Set is not implemented yet, as no-one is using BinaryP.

This operation is intended to be the "easy to use" function for changing the value of a GlBinaryP object. Set corresponds to the assignment operators of the other Gl<xxx>P classes, except that it allows both of the attributes to be set (assignment won't do this). That is, the other parameter types have an overloaded assignment operator that allows you to change their values with an easy to use syntax, e.g.,: GlLongP mylong(27, "longnumber"); mylong = 14; But since GlBinaryP has two "values" (i.e., buffer and length), you can't use assignment to change it.

RETURNS GlBinaryP&, a reference to self

GLOBALS none

FILES none

END_PROLOG

- value

  `void* value() const;`

  copy from a GlBinaryP BEGIN_PROLOG

  NAME value() const

  DESCRIPTION

  This operation returns the base value of this object

  RETURNS void*, a pointer to this object's base value

  GLOBALS none

  FILES none

  END_PROLOG

4.3.18.3.1 GlCircleP Class

Overview:

The following classes are defined in this file: GlCircleP a spcialized parameter for circle.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlCircleP_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlCircleP(); default constructor GlCircleP(const GlCircleP&); copy constructor GlCircleP(constructor given center and const GlPoint& center, radius float radius, char *name = 0); virtual ~GlCircleP(); destructor

GlCircleP& operator=(assignment operator const GlCircleP&); GlCircleP& operator=(overloaded assignment operator char* templateStr); for converting from SQS string

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); virtual GlStatus LoadString(load from string stream istrstream& templateStr);

virtual RWCString asString() const; convert to string format virtual RWCString asTemplate() const; convert to SQS format virtual void asString(output in string format ostrstream& os) const; to output stream virtual void asTemplate(output in SQS format ostrstream& os) const; to output stream void saveOn(human-readable output to stream ostream &) const;

void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

GlPoint GetCenter() const; return center point float GetRadius() const; return radius

void SetCenter(reset center const GlPoint& c); void SetRadius(reset radius float r);

void Set(reset both center and radius const GlPoint& c, float r);

RWBoolean deepCompare(compare base value object with const GlParameter&) const; self object GlParameter* NewSpecies() const; create an instance of same class void deepAssign(make a deep copy of elf object const GlParameter&);

Export Control: Public

Inheritance Relationships:

Inherits from GlSpatialP

Attributes:

`myCenter: GlPoint`

`myRadius: float`

center point

Constructors and Destructor:

```
GlCircleP(const GlPoint& c, float r, char* name);
```

copy constructor

```
GlCircleP(const GlCircleP& rhs);
```

default constructor

```
GlCircleP();
```

BEGIN_PROLOG

NAME GlCircleP()

DESCRIPTION

default constructor, required to have this a RWCollectable

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

```
virtual  ~GlCircleP();
```

BEGIN_PROLOG

NAME ~GlCircleP()

DESCRIPTION

Destructor.

RETURNS nothing

GLOBALS none

FILES none

Operations:

- asString

```
virtual void asString(ostrstream& os) const;
```

convert to SQS format BEGIN_PROLOG

NAME asString(ostrstream& os) const

DESCRIPTION

This operation writes the value of this object in string format to the output stream. The output format is "(X1, Y1, R)".

RETURNS RWCString.

GLOBALS none

FILES none

END_PROLOG

- asString

```
virtual RWCString asString() const;
```

BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. The output format is "(X1, Y1, R)".

RETURNS RWCString.

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
virtual RWCString asTemplate() const;
```

convert to string format BEGIN_PROLOG

NAME asTemplate() const

DESCRIPTION

This operation returns the value of this object in SQS format. The output format is "Circle (X1, Y1, R)".

RETURNS RWCString

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
virtual void asTemplate(ostrstream& os) const;
```

to output stream BEGIN_PROLOG

NAME asTemplate(ostrstream& os) const

DESCRIPTION

This operation writes the value of this object in SQS format to output stream. The output format is "CIRCLE (X1, Y1, R)".

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- copy

  ```
  virtual RWCollectable* copy() const;
  ```

- deepAssign

  ```
  void deepAssign(const GlParameter& rhs);
  ```

  create an instance of same class BEGIN_PROLOG

  NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

  DESCRIPTION

  This operation transfers the inherited and specialized portions of the right-hand-side object to this object, if the objects don't already match.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- deepCompare

  ```
  int deepCompare(const GlParameter& rhs) const;
  ```

  BEGIN_PROLOG

  NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

  DESCRIPTION

  This operation performs a value comparison between the base values of the two objects.

  RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

  GLOBALS none

  FILES none

  END_PROLOG

- GetCenter

  ```
  GlPoint GetCenter() const;
  ```

  RW-standard file BEGIN_PROLOG

  NAME GetCenter()

  DESCRIPTION

  This operation returns the center point for the circle.

  RETURNS GlPoint

  GLOBALS none

  FILES none

END_PROLOG

- GetRadius

```
float GetRadius() const;
```

return center point BEGIN_PROLOG

NAME GetRadius()

DESCRIPTION

This operation returns the radius value for the circle.

RETURNS float

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr);
```

BEGIN_PROLOG

NAME LoadString(istrstream& templateStr)

DESCRIPTION

This operation reads the string format from input stream and sets the valus of the objects accordingly. Expected format is "(X1, Y1, R)".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE if any conversion failed.

GLOBALS none

FILES none

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

for converting from SQS string BEGIN_PROLOG

NAME LoadTemplate(istrstream& templateStr)

DESCRIPTION

This operation reads the SQS format from input stream and set the valus of the objects accordingly. Expected format is "CIRCLE (X1, Y1, R)".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- NewSpecies

```
GlParameter* NewSpecies() const;
```

self object BEGIN_PROLOG

NAME NewSpecies() const

DESCRIPTION

This operation creates a new instance of a GlCircleP, but returns the pointer to the new object as a (generic) GlParameter*.

RETURNS GlParameter*, the pointer to the newly allocated object

GLOBALS none

FILES none

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- operator =

```
GlCircleP& operator =(const GlCircleP& rhs);
```

destructor BEGIN_PROLOG

NAME operator=(const const GlCircleP& other) // right hand side

DESCRIPTION

This operation copies the inherited and specialized information from the right-hand-side object if this object and the other one don't already match.

RETURNS GlCircleP&, reference to self

GLOBALS none

FILES none

- operator =

```
GlCircleP& operator =(char* templateStr);
```

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

RW-standard file NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

RW-standard stream BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

RW-standard stream NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

to output stream BEGIN_PROLOG

NAME saveOn(ostream os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a output stream (human readable format).

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetCenter

```
void SetCenter(const GlPoint& c);
```

return radius BEGIN_PROLOG

NAME SetCenter(const GlPoint& c)

DESCRIPTION

This operation resets the center point, and set valid flag accordingly depending on the value.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetRadius

```
void SetRadius(float r);
```

BEGIN_PROLOG

NAME SetRadius(float r)

DESCRIPTION

This operation resets the radius for the circle, and set valid flag accordingly depending on the value.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- Set

```
void Set(const GlPoint& c, float r);
```

BEGIN_PROLOG

NAME Set(const GlPoint& c, float r)

DESCRIPTION

This operation resets both center and radius for the circle, and set valid flag accordingly depending on the value.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.2 GlDateP Class

Overview:

The following classes are defined in this file: GlDateP specialization of GlParameter for dates

The following const variables are defined in this file: none

The following defines are defined in this file: GlDatePH make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlDateP(); default constructor needed for RWCollectable GlDateP(const GlDateP&); copy constructor GlDateP(const RWDate& d, normal constructor, parms = const char* name = 0); value and name virtual ~GlDateP(); destructor GlDateP& operator=(assignment operator const GlDateP&); copy from a GlDateP RWDate& operator=(assignment operator const RWDate&); copy from a RWDate const RWDate & value() const; retrieve parameter's value virtual RWCString asString() const; convert to string void saveOn(human-readable output ostream &) const; stream on which to write void saveGuts(XDR, etc. output RWvostream&) const; stream on which to write void saveGuts(XDR, etc. output RWFile&) const; file to which to write void restoreGuts(XDR, etc. input RWvistream&); stream from which to read void restoreGuts(XDR, etc. input RWFile&); file from which to read

RWBoolean deepCompare(compare base value const GlParameter&) const; object with which to compare GlParameter* NewSpecies()const; create new GlDateP pointed to as a GlParameterP void deepAssign(copy including base value const GlParameter&); generic parameter to copy RWspace binaryStoreSize() const; calculate store size including base value size

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

```
myDateBase: RWDate
```

Constructors and Destructor:

```
GlDateP(const RWDate& d, const char* name);
```

copy constructor

```
GlDateP(const GlDateP& rhs);
```

default constructor // needed for RWCollectable

```
GlDateP();
```

```
virtual  ~GlDateP();
```

value and name

Operations:

- asString

```
virtual RWCString asString() const;
```

retrieve parameter's value BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. This operation is implemented as a call to the RWDate asString method.

RETURNS RWCString, the output from the value() operation.

GLOBALS none

FILES none

END_PROLOG

- binaryStoreSize

```
unsigned long binaryStoreSize() const;
```

generic parameter to copy BEGIN_PROLOG

NAME binaryStoreSize() const

DESCRIPTION

This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

create new GlDateP pointed // to as a GlParameterP BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, IFF the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

required by rogue wave added by jie-bing yu BEGIN_PROLOG

NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

DESCRIPTION

This operation performs a value comparison between the base values of the two objects.

RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

```
GlParameter* NewSpecies() const;
```

object with which to compare BEGIN_PROLOG

NAME NewSpecies() const

DESCRIPTION

This operation creates a new instance of a GlDateP, but returns the pointer to the new object as a (generic) GlParameter*.

RETURNS GlParameter*, the pointer to the newly allocated GlDateP

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlDateP& operator =(const GlDateP& rhs);
```

destructor BEGIN_PROLOG

NAME operator=(const GlDateP& rhs) // right hand side

DESCRIPTION

This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

RETURNS GlDateP&, reference to self

GLOBALS none

FILES none

END_PROLOG

- operator =

```
RWDate& operator =(const RWDate& d);
```

copy from a GlDateP BEGIN_PROLOG

NAME operator=(const RWDate& d) // value to store

DESCRIPTION

assignment operator, stores the given date as the value of this object

RETURNS RWDate&, a reference this object's base value

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

file to which to write BEGIN_PROLOG

NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& file);
```

stream from which to read BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& file) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

convert to string BEGIN_PROLOG

NAME saveOn(ostream &os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a normal stream in a human-readable format

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- value

```
const RWDate& value() const;
```

copy from a RWDate BEGIN_PROLOG

NAME value() const

DESCRIPTION

This operation returns the base value of this object

RETURNS RWDate&, a reference to this object's base value

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.3 GlDoubleP Class

Overview:

The following classes are defined in this file: GlDoubleP specialization of GlParameter for doubles

The following const variables are defined in this file: none

The following defines are defined in this file: GlDoublePH make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlDoubleP(); default constructor needed for RWCollectable GlDoubleP(const GlDoubleP&); copy constructor GlDoubleP(double d, normal constructor, parms = const char* name = 0); value and name virtual ~GlDoubleP(); destructor GlDoubleP& operator=(assignment operator const GlDoubleP&); copy from a GlDoubleP double& operator=(assignment operator double); copy from a double double value() const; retrieve parameter's value virtual RWCString asString() const; convert to string void saveOn(human-readable output ostream &) const; stream on which to write void saveGuts(XDR, etc. output RWvostream&) const; stream on which to write void saveGuts(XDR, etc. output RWFile&) const; file to which to write void restoreGuts(XDR, etc. input RWvistream&); stream from which to read void restoreGuts(XDR, etc. input RWFile&); file from which to read RWBoolean deepCompare(compare base value const GlParameter&) const; object with which to compare GlParameter* NewSpecies()const; create new GlDoubleP pointed to as a GlParameterP void deepAssign(copy including base value const GlParameter&); generic parameter to copy RWspace binaryStoreSize() const; calculate store size including base value size

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

```
myDoubleBase: double
```

Constructors and Destructor:

```
GlDoubleP(double d, const char* name);
```

copy constructor

```
GlDoubleP(const GlDoubleP& rhs);
```

default constructor // needed for RWCollectable

```
GlDoubleP();
```

```
virtual  ~GlDoubleP();
```

value and name

Operations:

- asString

```
virtual RWCString asString() const;
```

retrieve parameter's value BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. This operation is implemented by streaming the value of this object onto a strstream, and converting that to a RWCString.

RETURNS RWCString, the output from the value() operation.

GLOBALS none

FILES none

END_PROLOG

return RWCString containing value

- binaryStoreSize

```
unsigned long binaryStoreSize() const;
```

generic parameter to copy BEGIN_PROLOG

NAME binaryStoreSize() const

DESCRIPTION

This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

create new GlDoubleP pointed // to as a GlParameterP BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, IFF the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

required by rogue wave added by jie-bing yu BEGIN_PROLOG

NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

DESCRIPTION

This operation performs a value comparison between the base values of the two objects.

RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

  ```
  virtual RWCollectable* newSpecies() const;
  ```

- NewSpecies

  ```
  GlParameter* NewSpecies() const;
  ```

  object with which to compare BEGIN_PROLOG

  NAME NewSpecies() const

  DESCRIPTION

  This operation creates a new instance of a GlDoubleP, but returns the pointer to the new object as a (generic) GlParameter*.

  RETURNS GlParameter*, the pointer to the newly allocated GlDoubleP

  GLOBALS none

  FILES none

  END_PROLOG

- operator =

  ```
  GlDoubleP& operator =(const GlDoubleP& rhs);
  ```

  destructor BEGIN_PROLOG

  NAME operator=(const GlDoubleP& rhs) // right hand side

  DESCRIPTION

  This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

  RETURNS GlDoubleP&, reference to self

  GLOBALS none

  FILES none

  END_PROLOG

- operator =

  ```
  double& operator =(double d);
  ```

  copy from a GlDoubleP BEGIN_PROLOG

  NAME operator=(double d) // value to store

  DESCRIPTION

  assignment operator, stores the given double as the value of this object

  RETURNS double&, a reference this object's base value

  GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

file to which to write BEGIN_PROLOG

NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& file);
```

stream from which to read BEGIN_PROLOG

NAME restoreGuts(RWFile& f) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

  `void saveGuts(RWFile& file) const;`

  stream on which to write BEGIN_PROLOG

  NAME saveGuts(RWFile& f) const // file to which to write

  DESCRIPTION

  This operation outputs the information from this object to a rogue wave file in RW-standard file format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- saveOn

  `void saveOn(ostream& os) const;`

  convert to string BEGIN_PROLOG

  NAME saveOn(ostream &os) const // stream on which to write

  DESCRIPTION

  This operation outputs the information from this object to a normal stream in a human-readable format

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- value

  `double value() const;`

  copy from a double BEGIN_PROLOG

  NAME value() const

  DESCRIPTION

  This operation returns the base value of this object

  RETURNS double, this object's base value

  GLOBALS none

  FILES none

  END_PROLOG

4.3.18.3.4 GlGPolygonP Class

Overview:

The following classes are defined in this file: GlGPolygonP

The following const variables are defined in this file: none

The following defines are defined in this file: _GlGPolygonPH_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: void GlGPolygonP(); needed for RWCollectable void GlGPolygonP(const GlGPolygonP&); virtual ~GlGPolygonP();

GlGPolygonP& operator=(const GlGPolygonP&); GlGPolygonP& operator=(char* templateStr);

virtual GlStatus LoadTemplate(istrstream& templateStr); virtual GlStatus LoadString(istrstream& templateStr);

virtual RWCString asString() const; virtual RWCString asTemplate() const; virtual void asString(ostrstream& os) const; virtual void asTemplate(ostrstream& os) const;

size_t GetNumExclusionZone() const; GlStatus AddExclusionZone(const GlPolygon& aPolygon); GlStatus SetBoundary(const GlPolygon& aPolygon); const GlPolygon* GetBoundary() const; const GlPolygon* GetExclusionZone(int index) const; virtual void saveOn(ostream &) const; human-readable output

virtual void saveGuts(RWvostream&) const; XDR, etc. output virtual void saveGuts(RWFile&) const; virtual void restoreGuts(RWvistream&); XDR, etc. input virtual void restoreGuts(RWFile&);

RWBoolean deepCompare(const GlParameter&) const; GlParameter* NewSpecies() const; void deepAssign(const GlParameter&);

Export Control: Public

Inheritance Relationships:

Inherits from GlSpatialP

Attributes:

`myBoundary: GlPolygon*`

myExclusionZones: GlPolygonSet

Constructors and Destructor:

GlGPolygonP();

---------- BEGIN_PROLOG

NAME GlPolygonP::GlGPolygonP

DESCRIPTION Standard constructor.

RETURNS void

Globals None

Files None

END_PROLOG

GlGPolygonP(const GlGPolygonP& rhs);

needed for RWCollectable

virtual ~GlGPolygonP();

---------- BEGIN_PROLOG

NAME GlGPolygonP::~GlGPolygonP

DESCRIPTION Destructor

RETURNS void

Globals None

Files None

END_PROLOG

Operations:

- AddExclusionZone

GlStatus AddExclusionZone(const GlPolygon& aPolygon);

---------- BEGIN_PROLOG

NAME GlGPolygonP::AddExclusionZone

DESCRIPTION This method adds the exclusion zone argument (aPolygon) to the exclusion zones member attribute myExclusionZones.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- asString

  `virtual RWCString asString() const;`

  ---------- BEGIN_PROLOG

  NAME GlGPolygonP::asString

  DESCRIPTION This member outputs the components of the GlGPolygon to the returned RWCString.

  RETURNS RWCString

  Globals None

  Files None

  END_PROLOG

- asString

  `virtual void asString(ostrstream& os) const;`

  ---------- BEGIN_PROLOG

  NAME GlGPolygonP::asString

  DESCRIPTION This member outputs the components of the GlGPolygonP to the passed in ostrstream argument.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- asTemplate

  `virtual RWCString asTemplate() const;`

  ---------- BEGIN_PROLOG

  NAME GlGPolygonP::asTemplate

  DESCRIPTION This member outputs the ""gpolygon" keyword and the components of the GlGPolygonP to the returned RWCString.

  RETURNS RWCString

  Globals None

  Files None

  END_PROLOG

- asTemplate

  `virtual void asTemplate(ostrstream& os) const;`

  ---------- BEGIN_PROLOG

  NAME GlGPolygonP::asTemplate

DESCRIPTION This member outputs the ""gpolygon" keyword and the components of the GlGPolygonP to the passed in ostrstream argument.

RETURNS RWCString

Globals None

Files None

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

NAME GlGPolygonP::deepAssign

DESCRIPTION This method assigns it's boundry and exclusion polygons to those of the passed in GlGPolygonP.

RETURNS void

Globals None

Files None

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::deepCompare

DESCRIPTION This method compares the components of self to that of the passed in GlPolygonP argument and returns whether or not they are equal.

RETURNS RWBoolean TRUE if the GlGPolygonP argument equals self FALSE if the GlGPolygonP argument doesn't equal self

Globals None

Files None

END_PROLOG

- GetBoundary

```
const GlPolygon* GetBoundary() const;
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::GetBoundary

DESCRIPTION This method returns the address of the boundry polygon (myBoundary).

RETURNS A GlPolygon* containing the address of the boundary GlPolygon.

Globals None

Files None

END_PROLOG

- GetExclusionZone

```
const GlPolygon* GetExclusionZone(int index) const;
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::GetExclusionZone

DESCRIPTION This method returns the address of the exclusion GlPolygon indicated by the index argument.

RETURNS The address of the exclusion GlPolygon.

Globals None

Files None

END_PROLOG

- GetNumExclusionZone

```
unsigned int GetNumExclusionZone() const;
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::GetNumExclusionZone

DESCRIPTION This method returns the number of exclusion zones in the GlGPolygonP. the index argument.

RETURNS The size_t number of exclustion zones.

Globals None

Files None

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr);
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::LoadString

DESCRIPTION This member loads the the components of the GlGPolygon using the passed in istrstream argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::LoadTemplate

DESCRIPTION This member first validates the stream for containing the "gpolygon" keyword and then loads the the components of the GlGPolygon using the passed in istrstream argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- NewSpecies

```
GlParameter* NewSpecies() const;
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::NewSpecies

DESCRIPTION This method creates a new GlPolygonP.

RETURNS A GlParameterP pointer containing the address of the new GlGPolygonP.

Globals None

Files None

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- operator =

```
GlGPolygonP& operator =(const GlGPolygonP& rhs);
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::operator=

DESCRIPTION Assignment operator.

RETURNS reference to self

Globals None

Files None

END_PROLOG

- operator =

```
GlGPolygonP& operator =(char* templateStr);
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::LoadTemplate

DESCRIPTION Assignment operator taking a template string argument.

RETURNS A reference to self.

Globals None

Files None

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

NAME GlGPolygonP::restoreGuts

DESCRIPTION This method restores the contents of the object from the passed in RWvistream argument. It is typically used to restore the object from transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

XDR, etc. input NAME GlGPolygonP::restoreGuts

DESCRIPTION This method restores the contents of the object from the passed in RWFile argument. It is typically used to restore the object from transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

human-readable output BEGIN_PROLOG

NAME GlPolygonP::saveGuts

DESCRIPTION This method writes the contents of the object to the passed in stream argument. It is typically used to prepare the object for transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

XDR, etc. output BEGIN_PROLOG

NAME GlPolygonP::saveGuts

DESCRIPTION This method writes the contents of the object to the passed in RWFile argument. It is typically used to prepare the object for transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::saveOn

DESCRIPTION Writes text version of object to the passed in ostream for debugging use.

RETURNS void

Globals None

Files None

END_PROLOG

- SetBoundary

```
GlStatus SetBoundary(const GlPolygon& aPolygon);
```

---------- BEGIN_PROLOG

NAME GlGPolygonP::SetBoundary

DESCRIPTION This method sets the boundry polygon (myBoundary) to the value of the passed in GlPolygon argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

4.3.18.3.5 GlLogHelper Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`GlLogHelper();`

BEGIN_PROLOG

NAME GlLogHelper()

DESCRIPTION

default constructor, creates empty object

RETURNS nothing

END_PROLOG

`~GlLogHelper();`

BEGIN_PROLOG

NAME ~GlLogHelper()

DESCRIPTION

destructor, is called ONLY when the LAST GlLog instance is no longer in scope. This operation calls the GlLog::Cleanup routine, which deallocates the heap-storage variables in GlLog. Having the call to the cleanup in the destructor of this "helper" object (which is created as a static variable in the GlLog object) ensures that the cleanup routine will only be invoked when the LAST routine using a GlLog object is done. (Remember that GlLog is a singleton object used by multiple other objects.)

RETURNS nothing

END_PROLOG

Operations:

4.3.18.3.6 GlLog Class

Overview:

The following classes are defined in this file: GlLog place-holder for full-fledged log GlPfxstreambuf specialized streambuf to handle prefixes GlPfxstream specialized stream to handle prefixes

The following const variables are defined in this file: none

The following defines are defined in this file: GlLogH make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlLog: <no return> GlLog(); constructor NOTE: no destructor, uses the automatic destructor when goes out of scope static void Cleanup(); removes current log static ostream& Instance(); define static stream static void SetLog(define log file const RWCString& filename); log file name GlPfxstream: <no return> GlPfxstream(constructor const RWCString&, prefix text streambuf* output = cout.rdbuf()); default default = cout's <no return> ~GlPfxstream(); destructor

GlPfxstreambuf: <no return> GlPfxstreambuf(constructor const RWCString&, prefix streambuf*); stream on which to write virtual ~GlPfxstreambuf(); destructor virtual int overflow(auto-call end-of-buffer int ch = EOF); default char to pass

Export Control: Public

Inheritance Relationships:

Attributes:

myFile: ofstream*

myHelper: GlLogHelper

```
myInstance: ostream*
```

Constructors and Destructor:

```
GlLog();
```

BEGIN_PROLOG

NAME GlLog()

DESCRIPTION

default constructor

RETURNS nothing

END_PROLOG

Operations:

- Cleanup

```
static void Cleanup();
```

constructor BEGIN_PROLOG

NAME GlLog::Cleanup()

DESCRIPTION

This operation flushes any outstanding information to the currently defined buffer/file (whichever is in existence) and cleans up any heap storage associated with same.

RETURNS void

END_PROLOG

- Instance

```
static ostream& Instance();
```

BEGIN_PROLOG

NAME GlLog::Instance()

DESCRIPTION

This operation is a static function to initialize the static stream which implements the GlLog object.

RETURNS ostream&, reference to the stream on which GlLog entries will be written.
END_PROLOG

- SetLog

```
static void SetLog(const RWCString& filename);
```

define static stream BEGIN_PROLOG

NAME GlLog::SetLog(const RWCString& filename) // new log's filename

DESCRIPTION

This operation establishes a new file (and buffer stream) for the (singleton) GlLog object. This is used to redirect the GlLog from standard output to a file, or from one file to another file.

RETURNS void

END_PROLOG

4.3.18.3.7 GlLongP Class

Overview:

The following classes are defined in this file: GlLongP specialization of GlParameter for type long

The following const variables are defined in this file: none

The following defines are defined in this file: _GlLongPH_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlLongP(); default constructor needed for RWCollectable GlLongP(const GlLongP&); copy constructor GlLongP(long l, normal constructor, parms = const char* name = 0); value and name virtual ~GlLongP(); destructor GlLongP& operator=(assignment operator const GlLongP&); copy from a GlLongP long& operator=(assignment operator long); copy from a long long value() const; retrieve parameter's value virtual RWCString asString() const; convert to string void saveOn(human-readable output ostream &) const; stream on which to write void saveGuts(XDR, etc. output RWvostream&) const; stream on which to write void saveGuts(XDR, etc. output RWFile&) const; file to which to write void restoreGuts(XDR, etc. input RWvistream&); stream from which to read void restoreGuts(XDR, etc. input RWFile&); file from which to read RWBoolean deepCompare(compare base value const GlParameter&) const; object with which to compare GlParameter* NewSpecies()const; create new GlLongP pointed to as a GlParameterP void deepAssign(copy including base value const GlParameter&); generic parameter to copy RWspace binaryStoreSize() const; calculate store size including base value size

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

```
myLongBase: long
```

Constructors and Destructor:

```
GlLongP(long l, const char* name);
```

copy constructor

```
GlLongP(const GlLongP& rhs);
```

default constructor needed for RWCollectable

```
GlLongP();
```

```
virtual  ~GlLongP();
```

value and name

Operations:

- asString

```
virtual RWCString asString() const;
```

retrieve parameter's value BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. This is implemented by streaming the value into a character array and creating a RWCString out of the character array.

RETURNS RWCString, a rogue wave string containing the value of this object

GLOBALS none

FILES none

END_PROLOG

- binaryStoreSize

```
unsigned long binaryStoreSize() const;
```

generic parameter to copy BEGIN_PROLOG

NAME binaryStoreSize() const

DESCRIPTION

This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

create new GlLongP pointed // to as a GlParameterP BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, IFF the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

required by rogue wave added by jie-bing yu BEGIN_PROLOG

NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

DESCRIPTION

This operation performs a value comparison between the base values of the two objects.

RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

  ```
  GlParameter* NewSpecies() const;
  ```

  object with which to compare BEGIN_PROLOG

  NAME NewSpecies() const

  DESCRIPTION

  This operation creates a new instance of a GlLongP, but returns the pointer to the new object as a (generic) GlParameter*.

  RETURNS GlParameter*, the pointer to the newly allocated GlLongP

  GLOBALS none

  FILES none

  END_PROLOG

- operator =

  ```
  GlLongP& operator =(const GlLongP& rhs);
  ```

  destructor BEGIN_PROLOG

  NAME operator=(const GlLongP& rhs) // right hand side

  DESCRIPTION

  This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

  RETURNS GlStringP&, reference to self

  GLOBALS none

  FILES none

  END_PROLOG

- operator =

  ```
  long& operator =(long l);
  ```

  copy from a GlLongP BEGIN_PROLOG

  NAME operator=(long l) // value to store

  DESCRIPTION

  assignment operator, stores the given long as the value of this object

  RETURNS long&, a reference this object's base value

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWvistream& vis);`

  file to which to write BEGIN_PROLOG

  NAME restoreGuts(RWvistream& vis) // stream from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWFile& f);`

  stream from which to read BEGIN_PROLOG

  NAME restoreGuts(RWFile& f) // file from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave file in RW-standard file format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- saveGuts

  `void saveGuts(RWvostream& vos) const;`

  stream on which to write BEGIN_PROLOG

  NAME saveGuts(RWvostream& vos) const // stream on which to write

  DESCRIPTION

  This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWFile& f) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

convert to string BEGIN_PROLOG

NAME saveOn(ostream &os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a normal stream in a human-readable format

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- value

```
long value() const;
```

copy from a long BEGIN_PROLOG

NAME value() const

DESCRIPTION

This operation returns the base value of this object

RETURNS long, a reference to this object's base value

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.8 GlParameterBase Class

Overview:

This base class is the real RWCollectable, but since the RW macros don't allow pure virtual functions, GlParameter inherits from this and is not, in fact, a truly macro-ized RWCollectable. This is ok because no GlParameter object will ever be instantiated (only the specialized classes derived from this class will be), thus any question about saving and restoring and which type of object will be restored are moot.

Export Control: Public

Inheritance Relationships:

Attributes:

```
myDescription: RWCString
```

```
myName: RWCString
```

Constructors and Destructor:

```
GlParameterBase(const char* name);
```

```
GlParameterBase(const GlParameterBase& rhs);
```

name of this object BEGIN_PROLOG

NAME GlParameterBase(const GlParameterBase& rhs) // right-hand-side

DESCRIPTION

copy constructor, creates a new object from the provided object

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

```
virtual  ~GlParameterBase();
```

right hand side object

Operations:

- binaryStoreSize

  ```
  unsigned long binaryStoreSize() const;
  ```

  destructor BEGIN_PROLOG

  NAME binaryStoreSize() const

  DESCRIPTION

  This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

  RETURNS RWspace, per the rogue wave libraries

  GLOBALS none

  FILES none

  END_PROLOG

- copy

  ```
  virtual RWCollectable* copy() const;
  ```

- isA

  ```
  virtual unsigned short isA() const;
  ```

- newSpecies

  ```
  virtual RWCollectable* newSpecies() const;
  ```

- operator =

  ```
  GlParameterBase& operator =(const GlParameterBase& rhs);
  ```

  right hand side object BEGIN_PROLOG

  NAME operator=(const GlParameterBase& rhs) // right hand side

  DESCRIPTION

  This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

  RETURNS GlParameterBase&, reference to self

  GLOBALS none

  FILES none

  END_PROLOG

4.3.18.3.9 GlParameterList Class

Overview:

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

```
myParms: RWOrdered
```

Constructors and Destructor:

```
GlParameterList(const char* name);
```

stream from which to read

```
GlParameterList(const GlParameterList& rhs);
```

default constructor

```
GlParameterList(RWvistream& );
```

right hand side object note, the next one is not yet implemented.

```
GlParameterList();

virtual  ~GlParameterList();
```

name of object

Operations:

- apply

```
void apply(void (*) (class RWCollectable*, void*) ap, void* x);
```

- asString

```
virtual RWCString asString() const;
```

BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. This operation is implemented as a series of calls to the asString function of the collected members of this list.

RETURNS RWCString, the output from the value() operation.

GLOBALS none

FILES none

END_PROLOG

- at

```
GlParameter*& at(unsigned int i);
```

- at

```
const GlParameter* at(unsigned int i) const;
```

- binaryStoreSize

```
unsigned long binaryStoreSize() const;
```

estimate size BEGIN_PROLOG

NAME binaryStoreSize() const

DESCRIPTION

This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- clearAndDestroy

```
void clearAndDestroy();
```

- clear

```
void clear();
```

- contains

```
int contains(const GlParameter* p) const;
```

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

create new GlParameterList pointed to as a GlParameter BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, IFF the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

BEGIN_PROLOG

NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

DESCRIPTION

This operation performs a value comparison between all of the contained parameters in the two objects.

RETURNS RWBoolean, TRUE if all the contained objects in the two objects match, FALSE otherwise

GLOBALS none

FILES none

NOTES

This operation requires that all the contained objects also be in the identical order for this operation to evaluate to TRUE.

END_PROLOG

- DeepFindParameter

```
GlParameter* DeepFindParameter(const RWCString& name);
```

name of parameter to find BEGIN_PROLOG

NAME DeepFindParameter(const RWCString &name)

DESCRIPTION

This operation finds and returns a parameter by name no matter where it is in the parameterlist. That is, it does a recursive search

RETURNS GlParameter*, a pointer to the parameter of that name

GLOBALS none

FILES none

NOTES the RW find() requires an object that "isEqual" with the one sought.. this method is easier, but maybe slower...

END_PROLOG

- DeepFindParameter

```
const GlParameter* DeepFindParameter(const RWCString& name)
const;
```

name of parameter to find BEGIN_PROLOG

NAME DeepFindParameter(const RWCString &name) const

DESCRIPTION

This operation is a convenience function to the non-const version which allows for processing non-const inputs and outputs.

RETURNS const GlParameter*, a const pointer to the parameter of that name

GLOBALS none

FILES none

END_PROLOG

- doRecursiveDelete

```
virtual void doRecursiveDelete(RWTValOrderedVector<class
GlParameter*>& delobjs);
```

indent (prefix) character BEGIN_PROLOG

NAME doRecursiveDelete(RWTValOrderedVector<GlParameter*>&) // objects to delete

DESCRIPTION

This function is used by parameters that act as collections of parameters, e.g., GlParameterList, to recursively create a list of all parameters contained within them. For furthur explanation, see GlParameter::RecursiveDelete.

RETURNS void

GLOBALS none

FILES none

NOTES The delobjs vector is handled by function RecursiveDelete(), which is defined in GlParameter.h

END_PROLOG

- entries

```
unsigned int entries() const;
```

- EstimateXDRSize

```
long EstimateXDRSize() const;
```

return value in string format BEGIN_PROLOG

NAME EstimateXDRSize() const

DESCRIPTION

This operation estimates the amount of memory needed to store this object in XDR format

RETURNS long, the estimated length of this object

GLOBALS none

FILES none

NOTES Note: this is only tested for SUN machines

END_PROLOG

- FindParameter

```
GlParameter* FindParameter(const RWCString& name);
```

name of parameter to find BEGIN_PROLOG

NAME FindParameter(const RWCString &name)

DESCRIPTION

This operation finds and returns a parameter by name.

RETURNS GlParameter*, a pointer to the parameter of that name

GLOBALS none

FILES none

NOTES the RW find() requires an object that "isEqual" with the one sought.. this method is easier, but maybe slower...

END_PROLOG

- FindParameter

```
const GlParameter* FindParameter(const RWCString& name) const;
```

BEGIN_PROLOG

NAME FindParameter(const RWCString &name) const

DESCRIPTION

This operation is a convenience function to the non-const version which allows for processing non-const inputs and outputs.

RETURNS const GlParameter*, a const pointer to the parameter of that name

GLOBALS none

FILES none

END_PROLOG

- find

```
GlParameter* find(const GlParameter* p) const;
```

- first

```
GlParameter* first() const;
```

- FlattenToHeap

```
char* FlattenToHeap(long& actsize) const;
```

right hand side object BEGIN_PROLOG

NAME FlattenToHeap(int& actsize, const GlParameterList& theresults)

DESCRIPTION This operation outputs the information from this object to a file in standard (flattened) XDR format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- Flatten

```
virtual long Flatten(char* buffer, long buflen) const;
```

file to which to write

- Flatten

```
virtual void Flatten(const RWCString& fname) const;
```

file from which to read

- index

```
unsigned int index(const GlParameter* p) const;
```

- insertAt

```
void insertAt(unsigned int i, GlParameter* p);
```

- insert

```
GlParameter* insert(GlParameter* p);
```

- isA

```
virtual unsigned short isA() const;
```

- isEmpty

```
int isEmpty() const;
```

- last

```
GlParameter* last() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

```
GlParameter* NewSpecies() const;
```

right hand side object BEGIN_PROLOG

NAME NewSpecies() const

DESCRIPTION

This operation creates a new instance of a GlParameterList, but returns the pointer to the new object as a (generic) GlParameter*.

RETURNS GlParameter*, the pointer to the newly allocated GlParameterList

GLOBALS none

FILES none

END_PROLOG

- occurrencesOf

```
unsigned int occurrencesOf(const GlParameter* p) const;
```

- operator =

  `GlParameterList& operator =(const GlParameterList& rhs);`

  destructor BEGIN_PROLOG

  NAME operator=(const GlParameterList& rhs) // right hand side

  DESCRIPTION

  This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

  RETURNS GlParameterList&, reference to self

  GLOBALS none

  FILES none

  END_PROLOG

- operator []

  `GlParameter*& operator [](unsigned int i);`

  calculate size NOTE: The rest of the *PUBLIC* operations are STRAIGHT PASS-THROUGHS of RW functions (pass-through to myParms)

  BEGIN_PROLOG

  NAME Rogue Wave Pass Throughs

  DESCRIPTION

  The following operations are straight pass-throughs from Rogue Wave to myParms. They are not commented individually because that information is in the RogueWave manual.

  RETURNS various

  GLOBALS none

  FILES none

  END_PROLOG

- prepend

  `GlParameter* prepend(GlParameter* p);`

- removeAndDestroy

  `void removeAndDestroy(const GlParameter* p);`

- remove

  `GlParameter* remove(const GlParameter* p);`

- restoreGuts

  `void restoreGuts(RWvistream& vis);`

  file to which to write BEGIN_PROLOG

  NAME restoreGuts(RWvistream& vis) // stream from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWFile& file);`

  stream from which to read BEGIN_PROLOG

  NAME restoreGuts(RWFile& file) // file from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave file in RW-standard file format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- Restore

  `static void Restore(const RWCString& fname, GlParameterList*& obj);`

  length of memory buffer

- Restore

  `virtual long Restore(char* buffer, long buflen);`

  file from which to read

- Restore

  `virtual void Restore(const RWCString& fname);`

  length of memory buffer

- Restore

```
static long Restore(char* buffer, long buflen, GlParameterList*&
obj);
```

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& file) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

right hand side object BEGIN_PROLOG

NAME saveOn(ostream &os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a normal stream in a human-readable format

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- savePretty

  `virtual void savePretty(ostream& os, unsigned int level, char pfx) const;`

  next is neatly formatted output, as opposed to merely readable BEGIN_PROLOG

  NAME savePretty(ostream& os, // stream on which to write size_t level, // level of indenting char pfx) const // indent character

  DESCRIPTION

  This operation formats the outputting of (potentially recursive) parameter lists.

  RETURNS RWCString, the output from the value() operation.

  GLOBALS none

  FILES none

  END_PROLOG

4.3.18.3.10 GlParameter Class

Overview:

The following classes are defined in this file: GlParameterBase core of GlParameter, including attributes GlParameter base class for all parameter types

The following const variables are defined in this file: none

The following defines are defined in this file: GlParameterH make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file:

GlParameterBase:

GlParameterBase(default constructor const char* name = 0); name of this object GlParameterBase(copy constructor const GlParameterBase&); right hand side object GlParameterBase& operator=(assignment operator const GlParameterBase&); right hand side object virtual ~GlParameterBase(); destructor

RWspace binaryStoreSize() const; calculate store size

GlParameter:

GlParameter(normal constructor const char* name = 0); [opt] name of this object GlParameter(copy constructor const GlParameter&); right hand side object GlParameter& operator=(assignment operator const GlParameter&); right hand side object virtual ~GlParameter(); destructor void SetName(set name of this object const RWCString&); given text const RWCString& GetName() const; return name of this object void SetDescription(set description of this obj const RWCString&); given text const RWCString& GetDescription() const; return description virtual void saveOn(human-readable output ostream&) const = 0; stream to which to write virtual RWCString asString() const = 0; output in string format virtual RWCString asQuotedString() const; output quoted string void saveGuts(XDR, etc. output RWvostream&) const; stream on which to write void saveGuts(XDR, etc. output RWFile&) const; file to which to write void restoreGuts(XDR, etc. input RWvistream&); stream from which to read void restoreGuts(XDR, etc. input RWFile&); file from which to read virtual RWBoolean deepCompare(pure virtual version const GlParameter&) const = 0; object to which to compare virtual GlParameter* NewSpecies()const = 0; pure virtual version virtual void deepAssign(copy including base value const GlParameter&) = 0; generic parameter to copy virtual void RecursiveDelete(); destroy all in collection parameters RWspace binaryStoreSize() const; calculate store size including base value size RWBoolean operator==(equality operator const GlParameter&) const; right hand side object virtual GlParameter* deepCopy() const; copy including base values virtual void doRecursiveDelete(destroys heap attributes/ composite parameters RWTValOrderedVector<GlParameter*>&); list of objects to delete

declaration of GlParameter (to provide type-safe casting, add a virtual X* GetX(); for every derived type. (eventually, of course, RTTI will make all GetX() functions obsolete...)

Export Control: Public

Inheritance Relationships:

Inherits from GlParameterBase

Attributes:

Constructors and Destructor:

```
GlParameter(const char* name);
```

```
GlParameter(const GlParameter& rhs);
```

[opt] name of this object

```
virtual  ~GlParameter();
```

right hand side object //////////////////////////////// GlParameter ////////////////////////////////

BEGIN_PROLOG

NAME ~GlParameter()

DESCRIPTION

destructor, cleans up as necessary for this object prior to deallocation

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

destructor

Operations:

- asQuotedString

  ```
  virtual RWCString asQuotedString() const;
  ```

  output in string format BEGIN_PROLOG

  NAME asQuotedString() const

  DESCRIPTION

  This operation returns the value of this object in string format with quotes around the text. This operation is implemented using the asString function to provide the text to be quoted.

  RETURNS RWCString, the output from the asString operation with quotes around it.

  GLOBALS none

  FILES none

  END_PROLOG

- asString

  ```
  virtual RWCString asString() const = 0;
  ```

  stream to which to write

- binaryStoreSize

  ```
  unsigned long binaryStoreSize() const;
  ```

  destroy all in collection parameters BEGIN_PROLOG

  NAME binaryStoreSize() const

  DESCRIPTION

  This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion. There is no additional space required for a GlParameter, as it has no attributes other than inherited ones.

  RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- deepAssign

```
virtual void deepAssign(const GlParameter& ) = 0;
```

pure virtual version

- deepCompare

```
virtual int deepCompare(const GlParameter& ) const = 0;
```

file from which to read

- deepCopy

```
virtual GlParameter* deepCopy() const;
```

right hand side object BEGIN_PROLOG

NAME deepCopy()

DESCRIPTION

this operation performs a deep copy of the right-hand-side object, instead of merely copying references to the underlying data. This operation calls the NewSpecies operation (which is pure virtual) so as to create a new object of whatever type the "this" object is (i.e., could be any specialized class), then uses the (specialized) operation deepAssign to copy the data specific to the (specialized) object.

RETURNS GlParameter*, the newly (heap-allocated) object.

GLOBALS none

FILES none

END_PROLOG

- doRecursiveDelete

```
virtual void doRecursiveDelete(RWTValOrderedVector<class
GlParameter*>& );
```

BEGIN_PROLOG

NAME doRecursiveDelete(RWTValOrderedVector<GlParameter*>&) // objects to delete

DESCRIPTION

This function is used by parameters that act as collections of parameters, e.g., GlParameterList, to recursively create a list of all parameters contained within them. It is needed in this base class so that *any* GlParameter-derived class can be properly deleted in the situations where this function is needed. For furthur explanation, see GlParameter::RecursiveDelete.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- GetDescription

```
const RWCString& GetDescription() const;
```

given text BEGIN_PROLOG

NAME GetDescription()

DESCRIPTION

this operation returns the value of the description attribute of this object

RETURNS RWCString&, reference to the description attribute of this object

GLOBALS none

FILES none

END_PROLOG

- GetName

```
const RWCString& GetName() const;
```

given text BEGIN_PROLOG

NAME GetName()

DESCRIPTION

this operation returns the value of the name attribute of this object

RETURNS RWCString&, reference to the name attribute of this object

GLOBALS none

FILES none

END_PROLOG

- NewSpecies

```
virtual GlParameter* NewSpecies() const = 0;
```

object with which to compare

- operator ==

```
int operator ==(const GlParameter& other) const;
```

// calculate store size // including base value size BEGIN_PROLOG

NAME operator==(const GlParameter& other) const

DESCRIPTION

this operation indicates whether or not this object matches the right hand side object in terms of matching type, name, and description.

RETURNS RWBoolean, TRUE if the the two objects are the same, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlParameter& operator =(const GlParameter& rhs);
```

right hand side object BEGIN_PROLOG

NAME operator=(const GlParameter& rhs) // right hand side

DESCRIPTION

This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

RETURNS GlParameter&, reference to self

GLOBALS none

FILES none

END_PROLOG

- RecursiveDelete

```
virtual void RecursiveDelete();
```

generic parameter to copy BEGIN_PROLOG

NAME RecursiveDelete()

DESCRIPTION

This function should be used with care to recursively delete all parameters contained in a compound parameter, i.e., a parameter that contains other parameters. This is usually necessary when exiting a program that has parameters that were allocated off the heap, e.g., when using the Restore() functions. Be careful, however, because GlParameter*(s) make shallow copies when using the copy constructors and assignment operators. Thus, it would be possible to delete the same parameter several times if it's shared between GlParameterList(s). This function builds a list of objects to be deleted, automatically filtering out duplicates, then deletes the objects.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& file);
```

stream from which to read BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

file to which to write BEGIN_PROLOG

NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

output in quoted string BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& file) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const = 0;
```

return description BEGIN_PROLOG

NAME saveOn(ostream &os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a normal stream in a human-readable format

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetDescription

```
void SetDescription(const RWCString& description);
```

return name of this object BEGIN_PROLOG

NAME SetDescription(const RWCString &description)

DESCRIPTION

this operation sets the description attribute of this object

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetName

```
void SetName(const RWCString& name);
```

destructor BEGIN_PROLOG

NAME SetName(const RWCString &name)

DESCRIPTION

this operation sets the name attribute of this object

RETURNS void

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.11 GlParmType Class

Overview:

BEGIN_PROLOG

NAME GlAllP.h

DESCRIPTION

This header file provides the definitions which are required for all parameter types.

The following classes are defined in this file: none

The following const variables are defined in this file: none

The following defines are defined in this file: _GlAllPH_ // prevent this from being included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: PARAMETER PARAMETERLIST LONGP STRINGP DOUBLEP DATEP BINARYP TIMEP STATUS SPATIALP RECTANGLEP POINTP POINTSETP POLYGONP GPOLYGONP

The following functions are defined in this file: GlAllP() // required for restoreGuts operations

END_PROLOG

type identifiers, one for each parameter type the tag is a concession to the hp compiler bug

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.18.3.12 GlPfxstreambuf Class

Overview:

this class is defined in GlLog.cxx specialized streambuf that adds the start-of-line state and emits the given prefix at the beginning of each line

Export Control: Public

Inheritance Relationships:

Attributes:

`myPrefixFlag: int`

`myPrefix: RWCString`

write prefix or don't

`myStream: streambuf*`

what to put on each line

Constructors and Destructor:

`GlPfxstreambuf(const RWCString& prefix, streambuf* output);`

`virtual ~GlPfxstreambuf();`

stream on which to write BEGIN_PROLOG

NAME ~GlPfxstreambuf()

DESCRIPTION

destructor, cleans up as necessary for this object prior to deallocation

RETURNS nothing END_PROLOG

Operations:

- overflow

`virtual int overflow(int ch);`

destructor BEGIN_PROLOG

NAME GlPfxstreambuf::overflow(int ch) // next character to write

DESCRIPTION

This operation handles the addition of a prefix to the output stream when the buffer is full and an attempt to insert another character is made. Since this is a virtual function defined in streambuf class, it is called automatically when the put area of the streambuf is filled.

RETURNS int, the next character to write

END_PROLOG

4.3.18.3.13 GlPfxstream Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myBuffer: streambuf*
```

Constructors and Destructor:

```
GlPfxstream(const RWCString& prefix, streambuf* output);

 ~GlPfxstream();
```

default is cout's streambuf* BEGIN_PROLOG

NAME ~GlGlPfxstream()

DESCRIPTION

destructor, cleans up as necessary for this object prior to deallocation

RETURNS nothing

END_PROLOG

Operations:

4.3.18.3.14 GlPointP Class

Overview:

The following classes are defined in this file: GlPointP

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPointPH_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: void GlPointP(); needed for RWCollectable void GlPointP(const GlPointP&); void GlPointP(float lat, float lon, char *name = 0); virtual ~GlPointP();

GlPointP& operator=(const GlPointP&); GlPointP& operator=(char* templateStr); virtual GlStatus LoadTemplate(istrstream& templateStr); virtual GlStatus LoadString(istrstream& templateStr);

virtual RWCString asString() const; virtual RWCString asTemplate() const; virtual void asString(ostrstream& os) const; virtual void asTemplate(ostrstream& os) const; void saveOn(ostream &) const; human-readable output

void saveGuts(RWvostream&) const; XDR, etc. output void saveGuts(RWFile&) const; void restoreGuts(RWvistream&); XDR, etc. input void restoreGuts(RWFile&);

float GetLat() const; float GetLon() const; void SetLat(float lat); void SetLon(float lon); RWBoolean deepCompare(const GlParameter&) const; GlParameter* NewSpecies() const; void deepAssign(const GlParameter&);

Export Control: Public

Inheritance Relationships:

Inherits from GlSpatialP

Attributes:

```
myPoint: GlPoint
```

Constructors and Destructor:

```
GlPointP(float lat, float lon, char* name);
```

`GlPointP(const GlPointP& rhs);`

needed for RWCollectable

`GlPointP();`

---------- BEGIN_PROLOG

NAME GlPointP::GlPointP

DESCRIPTION Standard constructor.

RETURNS void

Globals None

Files None

END_PROLOG

`virtual  ~GlPointP();`

normal constructor ----------
BEGIN_PROLOG

NAME GlPointP::~GlPointP

DESCRIPTION Standard destructor, does nothing.

RETURNS void

Globals None

Files None

END_PROLOG

Operations:

- asString

  `virtual void asString(ostrstream& os) const;`

  ---------- BEGIN_PROLOG

  NAME GlPointP::asString

  DESCRIPTION This method writes the lat/lon contents of the GlPoint member attribute to the passed in stream argument. It is typically used to prepare the GlPointP for ASCII representation.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- asString

  `virtual RWCString asString() const;`

---------- BEGIN_PROLOG

NAME GlPointP::asString

DESCRIPTION This method returns the value contents of the GlPoint member attribute as a RWString. It is typically used to prepare the GlPointP for ASCII representation.

RETURNS RWCString

Globals None

Files None

END_PROLOG

- asTemplate

```
virtual RWCString asTemplate() const;
```

---------- BEGIN_PROLOG

NAME GlPointP::asTemplate

DESCRIPTION This method returns an ASCII version of the GlPointP in a RWCString including the "point" keyword.

RETURNS RWCString

Globals None

Files None

END_PROLOG

- asTemplate

```
virtual void asTemplate(ostrstream& os) const;
```

---------- BEGIN_PROLOG

NAME GlPointP::asTemplate

DESCRIPTION This method returns an ASCII version of the GlPointP in the passed in ostream argument.

RETURNS void, check the stream for status.

Globals None

Files None

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

---------- BEGIN_PROLOG

NAME GlPointP::deepAssign

DESCRIPTION This method assigns self the name, description and value of the passed in argument.

RETURNS void

Globals None

Files None

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

---------- BEGIN_PROLOG

NAME GlPointP::deepCompare

DESCRIPTION This method compares the values of the arument to self for equality.

RETURNS RWBoolean TRUE if equal FALSE if not equal

Globals None

Files None

END_PROLOG

- GetLat

```
float GetLat() const;
```

---------- BEGIN_PROLOG

NAME GlPointP::GetLat

DESCRIPTION Accessor method for the latitude.

RETURNS float latitude value

Globals None

Files None

END_PROLOG

- GetLon

```
float GetLon() const;
```

---------- BEGIN_PROLOG

NAME GlPointP::GetLon

DESCRIPTION Accessor method for the latitude.

RETURNS float longitude value

Globals None

Files None

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr);
```

---------- BEGIN_PROLOG

NAME GlPointP::LoadString

DESCRIPTION This method loads the value contents of the GlPoint member attribute from the passed in stream agrument. It first checks for the bounding parenthesis is being passed in. Upon completion it also unloads the bounding right parenthesis.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

---------- BEGIN_PROLOG

NAME GlPointP::LoadTemplate

DESCRIPTION This method loads the value contents of the GlPoint member attribute from the passed in stream agrument. It first checks for the "point" keyword is being passed in.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- NewSpecies

```
GlParameter* NewSpecies() const;
```

---------- BEGIN_PROLOG

NAME GlPointP::NewSpecies

DESCRIPTION This method allocates a new object of type GlPointP

RETURNS This method returns a pointer to a GlParameterP containing the address of the new GlPointP.

Globals None

Files None

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- operator =

```
GlPointP& operator =(const GlPointP& rhs);
```

---------- BEGIN_PROLOG

NAME GlPointP::operator=

DESCRIPTION Standard assignment operator.

RETURNS a reference to self after assignment

Globals None

Files None

END_PROLOG

- operator =

```
GlPointP& operator =(char* templateStr);
```

---------- BEGIN_PROLOG

NAME GlPointP::operator=

DESCRIPTION Standard assignment operator

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

---------- BEGIN_PROLOG

NAME GlPointP::restoreGuts

DESCRIPTION This method restores the contents of the GlPoint member attribute from the passed in stream argument. It is typically used to restore the GlPointP from transport (via file/stream) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

XDR, etc. input ---------- BEGIN_PROLOG

NAME GlPointP::restoreGuts

DESCRIPTION Alternate signature taking an explicit file reference.

RETURNS void

Globals None

Files None

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

human-readable output ---------- BEGIN_PROLOG

NAME GlPointP::saveGuts

DESCRIPTION This method writes the contents of the GlPoint member attribute to the passed in stream argument. It is typically used to prepare the GlPointP for transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

XDR, etc. output ---------- BEGIN_PROLOG

NAME GlPointP::saveGuts

DESCRIPTION Alternate signature taking an explict file reference.

RETURNS void

Globals None

Files None

END_PROLOG

- saveOn

  `void saveOn(ostream& os) const;`

  ---------- BEGIN_PROLOG

  NAME GlPointP::saveOn

  DESCRIPTION This method writes the value contents of the GlPoint member attribute to the passed in stream argument. It is typically used to prepare the GlPointP for ASCII representation for debugging purposes.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- SetLat

  `void SetLat(float lat);`

  ---------- BEGIN_PROLOG

  NAME GlPointP::SetLat

  DESCRIPTION Set method for the latitude.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- SetLon

  `void SetLon(float lon);`

  ---------- BEGIN_PROLOG

  NAME GlPointP::SetLon

  DESCRIPTION Set method for the longitude.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

4.3.18.3.15 GlPointSetP Class

Overview:

PointSet as GlParameter

Export Control: Public

Inheritance Relationships:

Inherits from GlSpatialP

Attributes:

```
myPointSet: GlPointSet
```

Constructors and Destructor:

```
GlPointSetP();
```

```
GlPointSetP(const GlPointSetP& rhs);
```

needed for RWCollectable

```
virtual  ~GlPointSetP();
```

destructor

Operations:

- AddPoint

```
virtual GlStatus AddPoint(const GlPoint& aPoint);
```

- asString

```
virtual void asString(ostrstream& os) const;
```

- asString

```
virtual RWCString asString() const;
```

- asTemplate

```
virtual void asTemplate(ostrstream& os) const;
```

- asTemplate

```
virtual RWCString asTemplate() const;
```

- copy

  ```
  virtual RWCollectable* copy() const;
  ```

- deepAssign

  ```
  virtual void deepAssign(const GlParameter& rhs);
  ```

- deepCompare

  ```
  virtual int deepCompare(const GlParameter& rhs) const;
  ```

- GetNumPoints

  ```
  unsigned int GetNumPoints() const;
  ```

- GetPoint

  ```
  const GlPoint* GetPoint(int index) const;
  ```

- InsertPoint

  ```
  virtual GlStatus InsertPoint(const GlPoint& aPoint, int index);
  ```

- isA

  ```
  virtual unsigned short isA() const;
  ```

- LoadString

  ```
  virtual GlStatus LoadString(istrstream& templateStr);
  ```

  convert from SQS template

- LoadTemplate

  ```
  virtual GlStatus LoadTemplate(istrstream& templateStr);
  ```

  convert from SQS template

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

```
virtual GlParameter* NewSpecies() const;
```

- operator =

```
GlPointSetP& operator =(const GlPointSetP& rhs);
```

assignment

- operator =

```
GlPointSetP& operator =(char* templateStr);
```

assignment

- RemovePoint

```
virtual GlStatus RemovePoint(int index);
```

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

XDR, etc. input RW-standard file input

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

RW-standard input

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

XDR, etc. output RW-standard file output

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

human-readable output RW-standard output

- saveOn

```
virtual void saveOn(ostream& os) const;
```

output to a normal stream, human-readable

4.3.18.3.16 GlPointSet Class

Overview:

The following classes are defined in this file: GlPointSet light weight data structure for a set of 2D points.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPointSet_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPointSet(); default constructor GlPointSet(const GlPointSet& other); default copy constructor virtual ~GlPointSet(); destructor

GlPointSet& operator=(assignment operator const GlPointSet& other); RWBoolean operator==(boolean equal operator const GlPointSet& other) const;

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); virtual GlStatus LoadString(load from string format istrstream& templateStr);

virtual RWCString asString() const; convert to string format virtual RWCString asTemplate() const; convert to SQS format

virtual void asString(output in string format ostrstream&) const; to output stream virtual void asTemplate(output in SQS format ostrstream&) const; to output stream

virtual RWBoolean IsValid() const; return valid status

virtual size_t GetNumPoints() const; return number of points virtual GlStatus AddPoint(const GlPoint& aPoint); add a point to the end of set virtual GlStatus InsertPoint(insert a point at a specific position const GlPoint& aPoint, int index); virtual GlStatus RemovePoint(remove a point at a specific position int index); const GlPoint* GetPoint(get the point at a specified position int index) const;

virtual void saveOn(save to a human-readable output ostream &) const;

virtual void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream virtual void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file virtual void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream virtual void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

Export Control: Public

Inheritance Relationships:

Attributes:

```
myPoints: RWTPtrOrderedVector<class GlPoint>
```

Constructors and Destructor:

```
GlPointSet();
```

BEGIN_PROLOG
NAME GlPointSet::GlPointSet()
DESCRIPTION
constructor, creates a new default object of GlPointSet class
RETURNS nothing
GLOBALS none
FILES none
END_PROLOG

```
GlPointSet(const GlPointSet& rhs);
```

default constructor BEGIN_PROLOG
NAME GlPointSet(const GlPointSet& other)
DESCRIPTION
default copy constructor, creates a new object from the input object
RETURNS nothing
GLOBALS none
FILES none
END_PROLOG

```
virtual  ~GlPointSet();
```

default copy constructor BEGIN_PROLOG
NAME ~GlPointSet()
DESCRIPTION
Destructor, clean up all the points.
RETURNS nothing
GLOBALS none
FILES none

Operations:

- AddPoint

  ```
  virtual GlStatus AddPoint(const GlPoint& aPoint);
  ```

  return number of points BEGIN_PROLOG

  NAME GlPointSet::AddPoint(const GlPoint& aPoint)

  DESCRIPTION

  This operation adds the input point at the end of the set.

  RETURNS GlStatus: GlStatus::SUCCESS if the point is valid, GlStatus::FAILURE if the point is not valid;

  GLOBALS none

  FILES none

  END_PROLOG

- asString

  ```
  virtual RWCString asString() const;
  ```

  BEGIN_PROLOG

  NAME asString() const

  DESCRIPTION

  This operation returns the value of this object in string format. The output format is "((X1, Y1), (X2, Y2), ...)".

  RETURNS RWCString.

  GLOBALS none

  FILES none

  END_PROLOG

- asString

  ```
  virtual void asString(ostrstream& os) const;
  ```

  convert to SQS format BEGIN_PROLOG

  NAME asString(ostrstream& os) const

  DESCRIPTION

  This operation writes the value of this object in string format to output stream. The output format is "((X1, Y1), (X2, Y2), ...)".

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- asTemplate

  `virtual void asTemplate(ostrstream& os) const;`

  to output stream BEGIN_PROLOG

  NAME asTemplate(ostrstream& os) const

  DESCRIPTION

  This operation writes the value of this object in SQS format to output stream. The output format is "((X1, Y1), (X2, Y2), ...)".

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- asTemplate

  `virtual RWCString asTemplate() const;`

  convert to string format BEGIN_PROLOG

  NAME asTemplate() const

  DESCRIPTION

  This operation returns the value of this object in SQS format. The output format is "((X1, Y1), (X2, Y2), ...)".

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- GetNumPoints

  `virtual unsigned int GetNumPoints() const;`

  return valid status BEGIN_PROLOG

  NAME GlPointSet::GetNumPoints()

  DESCRIPTION

  This operation returns number of points in the point set.

  RETURNS size_t, number of points.

  GLOBALS none

  FILES none

  END_PROLOG

- GetPoint

```
const GlPoint* GetPoint(int index) const;
```

BEGIN_PROLOG

NAME GlPointSet::GetPoint(int index) // index offset

DESCRIPTION

This operation returns point positioned at the specified index.

RETURNS GlPoint*, not NULL if such a point exist in point set.

GLOBALS none

FILES none

END_PROLOG

- InsertPoint

```
virtual GlStatus InsertPoint(const GlPoint& aPoint, int index);
```

add a point to the end of set BEGIN_PROLOG

NAME GlPointSet::InsertPoint(const GlPoint& aPoint, int index)

DESCRIPTION

This operation inserts a point at specified location.

RETURNS GlStatus: GlStatus::SUCCESS if the point is valid, GlStatus::FAILURE if the point is not valid;

GLOBALS none

FILES none

END_PROLOG

- IsValid

```
virtual int IsValid() const;
```

to output stream BEGIN_PROLOG

NAME GlPointSet::IsValid()

DESCRIPTION

check the validity of the point set.

RETURNS RWBoolean: TRUE if everything is valid (all points are valid) FALSE otherwise.

GLOBALS none

FILES none

END_PROLOG

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr);
```

BEGIN_PROLOG

NAME LoadString(istrstream& templateStr)

DESCRIPTION

This operation reads the string format from input stream and sets the valus of the objects accordingly. Expected format is "((X1, Y1), (X2, Y2))".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

BEGIN_PROLOG

NAME LoadTemplate(istrstream& templateStr)

DESCRIPTION

This operation reads the SQS format from input stream and set the valus of the objects accordingly. Expected format is "((X1, Y1), (X2, Y2))".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- operator ==

```
int operator ==(const GlPointSet& other) const;
```

BEGIN_PROLOG

NAME GlPointSet::operator==(const const GlPointSet& other) // right hand side

DESCRIPTION

This operation compares itself with the right-hand-side object.

RETURNS TRUE if all information for the two objects compared are the same. FALSE if the information for the two objects compared are different.

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlPointSet& operator =(const GlPointSet& rhs);
```

destructor BEGIN_PROLOG

NAME operator=(const const GlPointSet& other) // right hand side

DESCRIPTION

This operation copies the inherited and specialized information from the right-hand-side object if this object and the other one don't already match.

RETURNS GlPointSet&, reference to self

GLOBALS none

FILES none

- RemovePoint

```
virtual GlStatus RemovePoint(int index);
```

BEGIN_PROLOG

NAME GlPointSet::RemovePoint(int index)

DESCRIPTION

This operation removes a point at specified location.

RETURNS GlStatus: GlStatus::SUCCESS if the point is valid, GlStatus::FAILURE if the point is not valid;

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

RW-standard file NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

RW-standard stream BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

RW-standard stream NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

BEGIN_PROLOG

NAME saveOn(ostream os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a output stream (human readable format).

RETURNS void

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.17 GlPoint Class

Overview:

The following classes are defined in this file: GlPoint light weight data structure for spatial 2D point.

The following const variables are defined in this file: static const char GlCLeftParen = '('; left parenthesis static const char GlCRightParen = ')'; right parenthesis static const char GlCComma = ','; comma static const char GlCSpace = ' '; blank space

The following defines are defined in this file: _GlPoint_h_ make sure this isn't included twice

The following externs are defined in this file: extern RWBoolean GlFCharMatch(match a const character istrstream& is, char value, GlStatus& retc); extern RWBoolean GlFLiteralMatch(match a literal istrstream& is, char* value, GlStatus& retc);

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPoint(); defaul constructor GlPoint(float lat, float lon); constructor if point coordinates are known GlPoint(const GlPoint& other); default copy constructor ~GlPoint(); destructor

GlPoint& operator=(assignment operator const GlPoint& other); RWBoolean operator==(boolean equal operator const GlPoint& other) const;

RWBoolean IsValid() const; return valid status GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); GlStatus LoadString(load from string format istrstream& templateStr);

GlStatus SetPoint(reset point float lat, float lon); GlStatus SetLat(float lat); reset latitude GlStatus SetLon(float lon); reset longitude

float GetLat() const; return latitude float GetLon() const; return longitude

RWCString asString() const; convert to string format RWCString asTemplate() const; convert to SQS format

void asString(ostrstream& s) const; output in string format to stream void asTemplate(output in SQS format ostrstream& s) const; to output stream

void saveGuts(XDR, etc. output to RWvostream& vos) const; RW-standard stream void saveGuts(RWFile& f) const; XDR, etc. output to RW-standard file void restoreGuts(RWvistream&

vis); XDR, etc. input from RW-standard stream void restoreGuts(RWFile& f); XDR, etc. input from RW-standard file

The following classes are defined in this file: GlPointP

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPointPH_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: void GlPointP(); needed for RWCollectable void GlPointP(const GlPointP&); void GlPointP(float lat, float lon, char *name = 0); virtual ~GlPointP();

GlPointP& operator=(const GlPointP&); GlPointP& operator=(char* templateStr); virtual GlStatus LoadTemplate(istrstream& templateStr); virtual GlStatus LoadString(istrstream& templateStr);

virtual RWCString asString() const; virtual RWCString asTemplate() const; virtual void asString(ostrstream& os) const; virtual void asTemplate(ostrstream& os) const; void saveOn(ostream &) const; human-readable output

void saveGuts(RWvostream&) const; XDR, etc. output void saveGuts(RWFile&) const; void restoreGuts(RWvistream&); XDR, etc. input void restoreGuts(RWFile&);

float GetLat() const; float GetLon() const; void SetLat(float lat); void SetLon(float lon); RWBoolean deepCompare(const GlParameter&) const; GlParameter* NewSpecies() const; void deepAssign(const GlParameter&);

The following classes are defined in this file: GlPointSet light weight data structure for a set of 2D points.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPointSet_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPointSet(); default constructor GlPointSet(const GlPointSet& other); default copy constructor virtual ~GlPointSet(); destructor

GlPointSet& operator=(assignment operator const GlPointSet& other); RWBoolean operator==(boolean equal operator const GlPointSet& other) const;

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); virtual GlStatus LoadString(load from string format istrstream& templateStr);

virtual RWCString asString() const; convert to string format virtual RWCString asTemplate() const; convert to SQS format

virtual void asString(output in string format ostrstream&) const; to output stream virtual void asTemplate(output in SQS format ostrstream&) const; to output stream

virtual RWBoolean IsValid() const; return valid status

virtual size_t GetNumPoints() const; return number of points virtual GlStatus AddPoint(const GlPoint& aPoint); add a point to the end of set virtual GlStatus InsertPoint(insert a point at a specific position const GlPoint& aPoint, int index); virtual GlStatus RemovePoint(remove a point at a specific position int index); const GlPoint* GetPoint(get the point at a specified position int index) const;

virtual void saveOn(save to a human-readable output ostream &) const;

virtual void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream virtual void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file virtual void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream virtual void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

Export Control: Public

Inheritance Relationships:

Attributes:

`myLat: float`

`myLon: float`

latitude

Constructors and Destructor:

`GlPoint(const GlPoint& other);`

constructor if point coordinates are known BEGIN_PROLOG

NAME GlPoint(const GlPoint& other)

DESCRIPTION

constructor, creates a new object from the input object

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`GlPoint(float lat, float lon);`

default constructor BEGIN_PROLOG

NAME GlPoint(float lat, float lon)

DESCRIPTION

constructor, creates a new object from the given coordinates

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`GlPoint();`

longitude BEGIN_PROLOG

NAME GlPoint()

DESCRIPTION

constructor, creates a new default object of GlPoint class

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- asString

`void asString(ostrstream& os) const;`

convert to SQS format BEGIN_PROLOG

NAME asString(ostrstream& os) const

DESCRIPTION

This operation writes the value of this object in string format to output stream. The output format is "X, Y".

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- asString

```
RWCString asString() const;
```

return longitude BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. The output format is "X, Y".

RETURNS RWCString.

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
void asTemplate(ostrstream& os) const;
```

output in string format to output stream BEGIN_PROLOG

NAME asTemplate(ostrstream& os) const

DESCRIPTION

This operation writes the value of this object in SQS format to output stream. The output format is "(X, Y)".

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
RWCString asTemplate() const;
```

convert to string format BEGIN_PROLOG

NAME asTemplate() const

DESCRIPTION

This operation returns the value of this object in SQS format. The output format is "(X, Y)".

RETURNS RWCString.

GLOBALS none

FILES none

END_PROLOG

- GetLat

  `float GetLat() const;`

  reset longitude BEGIN_PROLOG

  NAME GetLat()

  DESCRIPTION

  This operation returns the latitude value for the point.

  RETURNS float, the latitude value

  GLOBALS none

  FILES none

  END_PROLOG

- GetLon

  `float GetLon() const;`

  return latitude BEGIN_PROLOG

  NAME GetLon()

  DESCRIPTION

  This operation returns the longitude value for the point.

  RETURNS float, the longitude value

  GLOBALS none

  FILES none

  END_PROLOG

- IsValid

  `int IsValid() const;`

  BEGIN_PROLOG

  NAME GlPoint::IsValid() const

  DESCRIPTION

  This operation checks the validity of the point. The valid range for latitude is -90.0 to 90.0, and valid range for longitude is -180.0 to 180.0 .

  RETURNS RWBoolean TRUE if all are valid, FALSE if one of them or all are invalid.

  GLOBALS none

  FILES none

  END_PROLOG

- LoadString

```
GlStatus LoadString(istrstream& templateStr);
```

BEGIN_PROLOG

NAME LoadString(istrstream& templateStr)

DESCRIPTION

This operation reads the string format from input stream and sets the valus of the objects accordingly. Expected format is "X, Y".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- LoadTemplate

```
GlStatus LoadTemplate(istrstream& templateStr);
```

return valid status BEGIN_PROLOG

NAME LoadTemplate(istrstream& templateStr)

DESCRIPTION

This operation reads the SQS format from input stream and set the valus of the objects accordingly. Expected format is "(X, Y)".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- operator ==

```
int operator ==(const GlPoint& other) const;
```

BEGIN_PROLOG

NAME operator==(const const GlPoint& other) // right hand side

DESCRIPTION

This operation compares itself with the right-hand-side object.

RETURNS TRUE if all information for the two objects compared are the same. FALSE if the information for the two objects compared are different.

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlPoint& operator =(const GlPoint& other);
```

default copy constructor BEGIN_PROLOG

NAME operator=(const const GlPoint& other) // right hand side

DESCRIPTION

This operation copies the inherited and specialized information from the right-hand-side object if this object and the other one don't already match.

RETURNS GlPoint&, reference to self

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

XDR, etc. output to RW-standard file BEGIN_PROLOG

NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

XDR, etc. input from RW-standard stream BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

RW-standard stream BEGIN_PROLOG

NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

to output stream BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetLat

```
GlStatus SetLat(float lat);
```

BEGIN_PROLOG

NAME SetLat(float lat) // latitude value

DESCRIPTION

This operation resets the latitude for the point.

RETURNS GlStatus, GlStatus::SUCCESS;

GLOBALS none

FILES none

END_PROLOG

- SetLon

```
GlStatus SetLon(float lon);
```

reset latitude BEGIN_PROLOG

NAME SetLon(float lon) // latitude value

DESCRIPTION

This operation resets the longitude for the point.

RETURNS GlStatus, GlStatus::SUCCESS;

GLOBALS none

FILES none

END_PROLOG

- SetPoint

```
GlStatus SetPoint(float lat, float lon);
```

BEGIN_PROLOG

NAME SetPoint(float lat, // latitude value float lon) // longitude value

DESCRIPTION

This operation resets the corrdinates for the point.

RETURNS GlStatus, GlStatus::SUCCESS;

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.18 GlPolygonP Class

Overview:

The following classes are defined in this file: GlPolygonP

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPolygonPH_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: void GlPolygonP(); needed for RWCollectable void GlPolygonP(const GlPolygonP&); virtual ~GlPolygonP();

GlPolygonP& operator=(const GlPolygonP&); GlPolygonP& operator=(char* templateStr);

virtual GlStatus LoadString(istrstream& is); virtual GlStatus LoadTemplate(istrstream& templateStr); virtual RWCString asString() const; virtual void asString(ostrstream& os) const; virtual RWCString asTemplate() const; virtual void asTemplate(ostrstream& os) const;

size_t GetNumPoints() const; virtual GlStatus AddPoint(const GlPoint& aPoint); virtual GlStatus InsertPoint(const GlPoint& aPoint, int index); virtual GlStatus RemovePoint(int index); const GlPoint* GetPoint(int index) const;

virtual void saveOn(ostream &) const; human-readable output

virtual void saveGuts(RWvostream&) const; XDR, etc. output virtual void saveGuts(RWFile&) const; virtual void restoreGuts(RWvistream&); XDR, etc. input virtual void restoreGuts(RWFile&);

GlParameter* NewSpecies() const; RWBoolean deepCompare(const GlParameter& rhs) const; void deepAssign(const GlParameter& rhs);

Export Control: Public

Inheritance Relationships:

Inherits from GlSpatialP

Attributes:

`myPolygon: GlPolygon`

Constructors and Destructor:

`GlPolygonP();`

---------- BEGIN_PROLOG

NAME GlPolygonP::GlPolygonP

DESCRIPTION Standard constructor.

RETURNS void

Globals None

Files None

END_PROLOG

`GlPolygonP(const GlPolygonP& rhs);`

needed for RWCollectable

`virtual  ~GlPolygonP();`

---------- BEGIN_PROLOG

NAME GlPolygonP::~GlPolygonP

DESCRIPTION Standard destructor, does nothing.

RETURNS void

Globals None

Files None

END_PROLOG

Operations:

- AddPoint

  ```
  virtual GlStatus AddPoint(const GlPoint& aPoint);
  ```

  NAME GlPolygonP::AddPoint

  DESCRIPTION This method adds the GlPoint argument to the contained GlPolygon.

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- asString

  ```
  virtual void asString(ostrstream& os) const;
  ```

  ---------- BEGIN_PROLOG

  NAME GlPolygonP::asString

  DESCRIPTION This member outputs the points of the GlPolygon to the passed in ostrstream argument.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- asString

  ```
  virtual RWCString asString() const;
  ```

  ---------- BEGIN_PROLOG

  NAME GlPolygonP::asString

  DESCRIPTION This member outputs the points of the GlPolygon to a RWCString.

  RETURNS RWCString

  Globals None

  Files None

  END_PROLOG

- asTemplate

  `virtual void asTemplate(ostrstream& os) const;`

  ---------- BEGIN_PROLOG

  NAME GlPolygonP::asTemplate

  DESCRIPTION This member returns the "polygon" keyword plus the points of the GlPolygon to the passed in ostrstream argument.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- asTemplate

  `virtual RWCString asTemplate() const;`

  ---------- BEGIN_PROLOG

  NAME GlPolygonP::asTemplate

  DESCRIPTION This member returns the "polygon" keyword plus the points of the GlPolygon to the in an RWCString.

  RETURNS RWCString

  Globals None

  Files None

  END_PROLOG

- copy

  `virtual RWCollectable* copy() const;`

- deepAssign

  `void deepAssign(const GlParameter& rhs);`

  ---------- BEGIN_PROLOG

  NAME GlPolygonP::deepAssign

  DESCRIPTION This method sets all components and subcomponents of self to the values held in the passed in argument.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- deepCompare

  `int deepCompare(const GlParameter& rhs) const;`

  ---------- BEGIN_PROLOG

  NAME GlPolygonP::deepCompare

  DESCRIPTION This method compares the components of self to that of the passed in GlPolygonP argument and returns whether or not they are equal.

  RETURNS RWBoolean TRUE if the GlPolygonP argument equals self FALSE if the GlPolygonP argument doesn't equal self

  Globals None

  Files None

  END_PROLOG

- GetNumPoints

  `unsigned int GetNumPoints() const;`

  NAME GlPolygonP::GetNumPoints

  DESCRIPTION This method retrieves the number of points currently stored in the contained GlPolygon member attribute.

  RETURNS size_t

  Globals None

  Files None

  END_PROLOG

- GetPoint

  `const GlPoint* GetPoint(int index) const;`

  NAME GlPolygonP::GetPoint

  DESCRIPTION This method retrieves the address of the GlPoint at the position indicated by the index argument.

  RETURNS GlPoint*

  Globals None

  Files None

  END_PROLOG

- InsertPoint

  `virtual GlStatus InsertPoint(const GlPoint& aPoint, int index);`

  NAME GlPolygonP::InsertPoint

  DESCRIPTION This method adds the GlPoint argument to the contained GlPolygon at the position indicated by the index argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- LoadString

```
virtual GlStatus LoadString(istrstream& is);
```

---------- BEGIN_PROLOG

NAME GlPolygonP::LoadString

DESCRIPTION This member loads the the points of the GlPolygon using the passed in istrstream argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

---------- BEGIN_PROLOG

NAME GlPolygonP::LoadTemplate

DESCRIPTION This method loads the value contents of the GlPolygon member attribute from the passed in stream agrument. It first checks for that the polygon keyword is being passed in.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- NewSpecies

```
GlParameter* NewSpecies() const;
```

---------- BEGIN_PROLOG

NAME GlPolygonP::NewSpecies

DESCRIPTION This method creates a new GlPolygonP.

RETURNS A GlParameterP pointer containing the address of the new GlPolygonP.

Globals None

Files None

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- operator =

```
GlPolygonP& operator =(const GlPolygonP& rhs);
```

- operator =

```
GlPolygonP& operator =(char* templateStr);
```

---------- BEGIN_PROLOG

NAME GlPolygonP::operator=

DESCRIPTION Assignment operator taking a template string.

RETURNS a reference to self.

Globals None

Files None

END_PROLOG

- RemovePoint

```
virtual GlStatus RemovePoint(int index);
```

NAME GlPolygonP::RemovePoint

DESCRIPTION This method removes the GlPoint at the position indicated by the index argument from it's contained GlPolygon.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

XDR, etc. input ---------- BEGIN_PROLOG

NAME GlPolygonP::restoreGuts

DESCRIPTION This method restores the contents of the object from the passed in RWFile argument. It is typically used to restore the object from transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

---------- BEGIN_PROLOG

NAME GlPolygonP::restoreGuts

DESCRIPTION This method restores the contents of the object from the passed in RWvistream argument. It is typically used to restore the object from transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

XDR, etc. output ----------
BEGIN_PROLOG

NAME GlPolygonP::saveGuts

DESCRIPTION This method writes the contents of the object to the passed in file argument. It is typically used to prepare the object for transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

human-readable output ----------
BEGIN_PROLOG

NAME GlPolygonP::saveGuts

DESCRIPTION This method writes the contents of the object to the passed in stream argument. It is typically used to prepare the object for transport (via file) or persistent storage.

RETURNS void

Globals None

Files None

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

---------- BEGIN_PROLOG

NAME GlPolygonP::saveOn

DESCRIPTION Writes text version of object to the passed in ostream for debugging use.

RETURNS void

Globals None

Files None

END_PROLOG

4.3.18.3.19 GlPolygonSetP Class

Overview:

The following classes are defined in this file: GlPolygonSetP a spcialized parameter for polygon set.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPolygonP_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPolygonSetP(); default constructor GlPolygonSetP(const GlPolygonSetP&); copy constructor virtual ~GlPolygonSetP(); destructor

GlPolygonSetP& operator=(assignment operator const GlPolygonSetP&); GlPolygonSetP& operator=(overloaded assignment operator char* templateStr); for converting from SQS string

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); virtual GlStatus LoadString(load from string stream istrstream& templateStr);

virtual RWCString asString() const; convert to string format virtual RWCString asTemplate() const; convert to SQS format virtual void asString(output in string format ostrstream&) const; to output stream virtual void asTemplate(output in SQS format ostrstream&) const; to output stream

size_t GetNumPolygons() const; get number of polygons GlStatus AddPolygon(add polygon at end const GlPolygon& aPolygon); GlStatus InsertPolygon(insert polygon at specified const GlPolygon& aPolygon, position int index); GlStatus RemovePolygon(int index); remove polygon from specified const GlPolygon* GetPolygon(position int index) const; virtual void saveOn(human-readable output ostream &) const;

void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

RWDECLARE_COLLECTABLE(GlPolygonSetP);

RWBoolean deepCompare(compare base value object with const GlParameter&) const; self object GlParameter* NewSpecies() const; create an instance of same class void deepAssign(make a deep copy of self object const GlParameter&);

PolygonSet as GlParameter

Export Control: Public

Inheritance Relationships:

Inherits from GlSpatialP

Attributes:

`myPolygonSet: GlPolygonSet`

Constructors and Destructor:

`GlPolygonSetP();`

BEGIN_PROLOG

NAME GlPolygonSetP()

DESCRIPTION

default constructor, required to have this a RWCollectable

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`GlPolygonSetP(const GlPolygonSetP& rhs);`

default constructor

```
virtual  ~GlPolygonSetP();
```

copy constructor BEGIN_PROLOG

NAME ~GlPolygonSetP()

DESCRIPTION

Destructor, clean up all the polygons.

RETURNS nothing

GLOBALS none

FILES none

Operations:

- AddPolygon

```
GlStatus AddPolygon(const GlPolygon& aPolygon);
```

get number of polygons BEGIN_PROLOG

NAME GlPointSetP::AddPolygon(const GlPoint& aPolygon)

DESCRIPTION

This operation adds the input polygon at the end of the set.

RETURNS GlStatus: GlStatus::SUCCESS if the polygon is valid, GlStatus::FAILURE if the polygon is not valid;

GLOBALS none

FILES none

END_PROLOG

- asString

```
virtual void asString(ostrstream& os) const;
```

convert to SQS format BEGIN_PROLOG

NAME asString(ostrstream& os)

DESCRIPTION

This operation writes the value of this object in string format to output stream. The output format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)".

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- asString

```
virtual RWCString asString() const;
```

BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. The output format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)".

RETURNS RWCString.

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
virtual void asTemplate(ostrstream& os) const;
```

to output stream BEGIN_PROLOG

NAME asTemplate(ostrstream& os) const

DESCRIPTION

This operation writes the value of this object in SQS format to output stream. The output format is "POLYGON_SET (((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)".

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
virtual RWCString asTemplate() const;
```

convert to string format BEGIN_PROLOG

NAME asTemplate() const

DESCRIPTION

This operation returns the value of this object in SQS format. The output format is "POLYGON_SET (((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)".

RETURNS RWCString.

GLOBALS none

FILES none

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

  `void deepAssign(const GlParameter& rhs);`

  create an instance of same class BEGIN_PROLOG

  NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

  DESCRIPTION

  This operation transfers the inherited and specialized portions of the right-hand-side object to this object, if the objects don't already match.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- deepCompare

  `int deepCompare(const GlParameter& rhs) const;`

  BEGIN_PROLOG

  NAME GlPolygonSetP::deepCompare

  DESCRIPTION This method compares the values of the arument to self for equality.

  RETURNS RWBoolean TRUE if equal FALSE if not equal

  Globals None

  Files None

  END_PROLOG

- GetNumPolygons

  `unsigned int GetNumPolygons() const;`

  to output stream BEGIN_PROLOG

  NAME GlPointSetP::GetNumPoints()

  DESCRIPTION

  This operation returns number of polygons in the polygon set.

  RETURNS size_t, number of polygons.

  GLOBALS none

  FILES none

  END_PROLOG

- GetPolygon

  `const GlPolygon* GetPolygon(int index) const;`

  remove polygon from specified BEGIN_PROLOG

NAME GlPointSetP::GetPolygon(int index) // index offset

DESCRIPTION

This operation returns polygon positioned at the specified index.

RETURNS GlPolygon*, not NULL if such a point exist in point set.

GLOBALS none

FILES none

END_PROLOG

- InsertPolygon

```
GlStatus InsertPolygon(const GlPolygon& aPolygon, int index);
```

BEGIN_PROLOG

NAME GlPolygonSetP::InsertPolygon(const GlPolygon& aPolygon, int index)

DESCRIPTION

This operation inserts a polygon at specified location.

RETURNS GlStatus: GlStatus::SUCCESS if the polygon is valid, GlStatus::FAILURE if the polygon is not valid;

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr);
```

BEGIN_PROLOG

NAME LoadString(istrstream& templateStr)

DESCRIPTION

This operation reads the string format from input stream and sets the valus of the objects accordingly. Expected format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

for converting from SQS string BEGIN_PROLOG

NAME LoadTemplate(istrstream& templateStr)

DESCRIPTION

This operation reads the SQS format from input stream and sets the valus of the objects accordingly. Expected format is "POLYGON_SET (((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

```
GlParameter* NewSpecies() const;
```

self object BEGIN_PROLOG

NAME NewSpecies() const

DESCRIPTION

This operation creates a new instance of a GlPolygonSetP, but returns the pointer to the new object as a (generic) GlParameter*.

RETURNS GlParameter*, the pointer to the newly allocated object

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlPolygonSetP& operator =(const GlPolygonSetP& rhs);
```

destructor BEGIN_PROLOG

NAME operator=(const const GlPolygonSetP& other) // right hand side

DESCRIPTION

This operation copies the inherited and specialized information from the right-hand-side object if this object and the other one don't already match.

RETURNS GlPolygonSetP&, reference to self

GLOBALS none

FILES none

- operator =

```
GlPolygonSetP& operator =(char* templateStr);
```

BEGIN_PROLOG

NAME operator=(char* templateStr)

DESCRIPTION

This operation reads the SQS format from input stream and assign itself with values from the SQS stream. Expected format is "POLYGON_SET (((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)".

RETURNS none

GLOBALS none

FILES none

END_PROLOG

- RemovePolygon

```
GlStatus RemovePolygon(int index);
```

BEGIN_PROLOG

NAME GlPolygonSetP::RemovePolygon(int index)

DESCRIPTION

This operation removes a polygon at specified location. After removal, the postion of succeeding polygons will get moved forward one position.

RETURNS GlStatus: GlStatus::SUCCESS if index is valid, GlStatus::FAILURE if index is out of range.

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

RW-standard stream BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

RW-standard file NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

RW-standard stream NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

  `virtual void saveOn(ostream& os) const;`

  BEGIN_PROLOG

  NAME saveOn(ostream os) const // stream on which to write

  DESCRIPTION

  This operation outputs the information from this object to a output stream (human readable format).

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

4.3.18.3.20 GlPolygonSet Class

Overview:

The following classes are defined in this file: GlPolygonSet light weight data structure for a set of polygons.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPolygonSet_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPolygonSet(); default constructor GlPolygonSet(copy constructor const GlPolygonSet& other); virtual ~GlPolygonSet(); destructor

GlPolygonSet& operator=(assignment operator const GlPolygonSet& other); RWBoolean operator==(boolean equal operator const GlPolygonSet& other) const;

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr, RWBoolean partial=FALSE); virtual GlStatus LoadString(load from string format istrstream& templateStr, RWBoolean partial=FALSE);

virtual RWCString asString(convert to string format RWBoolean partial=FALSE) const; virtual RWCString asTemplate(convert to SQS format RWBoolean partial=FALSE) const; virtual void asString(output in string format ostrstream& os, to output stream RWBoolean partial=FALSE) const; virtual void asTemplate(output in SQS format ostrstream& os, to output stream RWBoolean partial=FALSE) const;

RWBoolean IsValid() const; return valid status void Empty(); empty the set of polygons size_t GetNumPolygons() const; return number of polygons GlStatus AddPolygon(add a polygon to the

end of set const GlPolygon& aPolygon); GlStatus InsertPolygon(insert a polygon at a specified const GlPolygon& aPolygon, position int index); GlStatus RemovePolygon(remove a point at a specified int index); position const GlPolygon* GetPolygon(get the polygon at a specified int index) const; position virtual void saveOn(save to a human-readable output ostream &) const;

virtual void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream virtual void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file virtual void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream virtual void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

The following classes are defined in this file: GlPolygonSetP a spcialized parameter for polygon set.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPolygonP_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPolygonSetP(); default constructor GlPolygonSetP(const GlPolygonSetP&); copy constructor virtual ~GlPolygonSetP(); destructor

GlPolygonSetP& operator=(assignment operator const GlPolygonSetP&); GlPolygonSetP& operator=(overloaded assignment operator char* templateStr); for converting from SQS string

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); virtual GlStatus LoadString(load from string stream istrstream& templateStr);

virtual RWCString asString() const; convert to string format virtual RWCString asTemplate() const; convert to SQS format virtual void asString(output in string format ostrstream&) const; to output stream virtual void asTemplate(output in SQS format ostrstream&) const; to output stream

size_t GetNumPolygons() const; get number of polygons GlStatus AddPolygon(add polygon at end const GlPolygon& aPolygon); GlStatus InsertPolygon(insert polygon at specified const GlPolygon& aPolygon, position int index); GlStatus RemovePolygon(int index); remove polygon from specified const GlPolygon* GetPolygon(position int index) const; virtual void saveOn(human-readable output ostream &) const;

void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

RWDECLARE_COLLECTABLE(GlPolygonSetP);

RWBoolean deepCompare(compare base value object with const GlParameter&) const; self object GlParameter* NewSpecies() const; create an instance of same class void deepAssign(make a deep copy of self object const GlParameter&);

Export Control: Public

Inheritance Relationships:

Attributes:

```
myPolygons: RWTPtrOrderedVector<class GlPolygon>
```

Constructors and Destructor:

```
GlPolygonSet();
```

BEGIN_PROLOG

NAME GlPolygonSet::GlPolygonSet()

DESCRIPTION

constructor, creates a new default object of GlPolygonSet class

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

```
GlPolygonSet(const GlPolygonSet& rhs);
```

default constructor BEGIN_PROLOG

NAME GlPolygonSet(const GlPolygonSet& other)

DESCRIPTION

copy constructor, creates a new object from the input object

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

```
virtual  ~GlPolygonSet();
```

BEGIN_PROLOG

NAME ~GlPolygonSet()

DESCRIPTION

Destructor, clean up all the polygons.

RETURNS nothing

GLOBALS none

FILES none

Operations:

- AddPolygon

  `GlStatus AddPolygon(const GlPolygon& aPolygon);`

  return number of polygons BEGIN_PROLOG

  NAME GlPolygonSet::AddPolygon(const GlPolygon& aPolygon)

  DESCRIPTION

  This operation adds the input polygon at the end of the set.

  RETURNS GlStatus: GlStatus::SUCCESS if the polygon is valid, GlStatus::FAILURE if the polygon is not valid;

  GLOBALS none

  FILES none

  END_PROLOG

- asString

  `virtual RWCString asString(int partial) const;`

  BEGIN_PROLOG

  NAME asString(RWBoolean partial) const // partial output

  DESCRIPTION

  This operation returns the value of this object in string format. The output format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)". IF partial is TRUE, then the first '(' is skipped.

  RETURNS RWCString.

  GLOBALS none

  FILES none

  END_PROLOG

- asString

  `virtual void asString(ostrstream& os, int partial) const;`

  BEGIN_PROLOG

  NAME asString(ostrstream& os, // output stream RWBoolean partial) const // partial output

DESCRIPTION

This operation writes the value of this object in string format to output stream. The output format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)". IF partial is TRUE, then the first '(' is skipped.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
virtual void asTemplate(ostrstream& os, int partial) const;
```

BEGIN_PROLOG

NAME asTemplate(ostrstream& os, RWBoolean partial) const

DESCRIPTION

This operation returns the value of this object in SQS format. The output format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)". IF partial is TRUE, then the first '(' is skipped.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- asTemplate

```
virtual RWCString asTemplate(int partial) const;
```

BEGIN_PROLOG

NAME asTemplate(RWBoolean partial) const // partial output

DESCRIPTION

This operation returns the value of this object in SQS format. The output format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)". IF partial is TRUE, then the first '(' is skipped.

RETURNS RWCString.

GLOBALS none

FILES none

END_PROLOG

- Empty

```
void Empty();
```

return valid status BEGIN_PROLOG

NAME GlPointSet::Empty()

DESCRIPTION

This operation empties the current set of polygons.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- GetNumPolygons

```
unsigned int GetNumPolygons() const;
```

empty the current set of polygons BEGIN_PROLOG

NAME GlPointSet::GetNumPoints()

DESCRIPTION

This operation returns number of polygons in the polygon set.

RETURNS size_t, number of polygons.

GLOBALS none

FILES none

END_PROLOG

- GetPolygon

```
const GlPolygon* GetPolygon(int index) const;
```

position BEGIN_PROLOG

NAME GlPointSet::GetPolygon(int index) // index offset

DESCRIPTION

This operation returns polygon positioned at the specified index.

RETURNS GlPolygon*, not NULL if such a point exist in point set.

GLOBALS none

FILES none

END_PROLOG

- InsertPolygon

```
GlStatus InsertPolygon(const GlPolygon& aPolygon, int index);
```

BEGIN_PROLOG

NAME GlPolygonSet::InsertPolygon(const GlPolygon& aPolygon, int index)

DESCRIPTION

This operation inserts a polygon at specified location.

RETURNS GlStatus: GlStatus::SUCCESS if the polygon is valid, GlStatus::FAILURE if the polygon is not valid;

GLOBALS none

FILES none

END_PROLOG

- IsValid

```
int IsValid() const;
```

BEGIN_PROLOG

NAME GlPolygonSet::IsValid() const

DESCRIPTION

This operation checks the validity of the polygon set.

RETURNS RWBoolean TRUE if all containging polygons are valid, FALSE if any one of them is invalid.

GLOBALS none

FILES none

END_PROLOG

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr, int
partial);
```

BEGIN_PROLOG

NAME LoadString(istrstream& templateStr, RWBoolean partial)

DESCRIPTION

This operation reads the string format from input stream and sets the valus of the objects accordingly. Expected format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)". If partial is TRUE, then the first '(' is ignored. For the case that empty polygon set is allowed.

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr, int
partial);
```

BEGIN_PROLOG

NAME LoadTemplate(istrstream& templateStr, RWBoolean partial)

DESCRIPTION

This operation reads the SQS format from input stream and sets the valus of the objects accordingly. Expected format is "(((X1, Y1), (X2, Y2), ...), ((X1, Y1), (X2, Y2), ...), ...)". If partial is TRUE, then the first '(' is ignored. For the case that empty polygon set is allowed.

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- operator ==

```
int operator ==(const GlPolygonSet& other) const;
```

BEGIN_PROLOG

NAME operator==()

DESCRIPTION

Equal operator.

RETURNS RWBoolean TRUE: if two objects are equal in value FALSE: if two objects are different in value

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlPolygonSet& operator =(const GlPolygonSet& rhs);
```

destructor BEGIN_PROLOG

NAME operator=(const const GlPolygonSet& other) // right hand side

DESCRIPTION

This operation copies the inherited and specialized information from the right-hand-side object if this object and the other one don't already match.

RETURNS GlPolygonSet&, reference to self

GLOBALS none

FILES none

- RemovePolygon

```
GlStatus RemovePolygon(int index);
```

BEGIN_PROLOG

NAME GlPolygonSet::RemovePolygon(int index)

DESCRIPTION

This operation removes a polygon at specified location. After removal, the postion of succeeding polygons will get moved forward one position.

RETURNS GlStatus: GlStatus::SUCCESS if index is valid, GlStatus::FAILURE if index is out of range.

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

RW-standard stream BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

RW-standard file NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

RW-standard stream NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

position BEGIN_PROLOG

NAME saveOn(ostream os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a output stream (human readable format).

RETURNS void

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.21 GlPolygon Class

Overview:

The following classes are defined in this file: GlPolygonP

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPolygonPH_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: void GlPolygonP(); needed for RWCollectable void GlPolygonP(const GlPolygonP&); virtual ~GlPolygonP();

GlPolygonP& operator=(const GlPolygonP&); GlPolygonP& operator=(char* templateStr);

virtual GlStatus LoadString(istrstream& is); virtual GlStatus LoadTemplate(istrstream& templateStr); virtual RWCString asString() const; virtual void asString(ostrstream& os) const; virtual RWCString asTemplate() const; virtual void asTemplate(ostrstream& os) const;

size_t GetNumPoints() const; virtual GlStatus AddPoint(const GlPoint& aPoint); virtual GlStatus InsertPoint(const GlPoint& aPoint, int index); virtual GlStatus RemovePoint(int index); const GlPoint* GetPoint(int index) const;

virtual void saveOn(ostream &) const; human-readable output

virtual void saveGuts(RWvostream&) const; XDR, etc. output virtual void saveGuts(RWFile&) const; virtual void restoreGuts(RWvistream&); XDR, etc. input virtual void restoreGuts(RWFile&);

GlParameter* NewSpecies() const; RWBoolean deepCompare(const GlParameter& rhs) const; void deepAssign(const GlParameter& rhs);

The following classes are defined in this file: GlPolygonSet light weight data structure for a set of polygons.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPolygonSet_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPolygonSet(); default constructor GlPolygonSet(copy constructor const GlPolygonSet& other); virtual ~GlPolygonSet(); destructor

GlPolygonSet& operator=(assignment operator const GlPolygonSet& other); RWBoolean operator==(boolean equal operator const GlPolygonSet& other) const;

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr, RWBoolean partial=FALSE); virtual GlStatus LoadString(load from string format istrstream& templateStr, RWBoolean partial=FALSE);

virtual RWCString asString(convert to string format RWBoolean partial=FALSE) const; virtual RWCString asTemplate(convert to SQS format RWBoolean partial=FALSE) const; virtual void asString(output in string format ostrstream& os, to output stream RWBoolean partial=FALSE)

const; virtual void asTemplate(output in SQS format ostrstream& os, to output stream RWBoolean partial=FALSE) const;

RWBoolean IsValid() const; return valid status void Empty(); empty the set of polygons size_t GetNumPolygons() const; return number of polygons GlStatus AddPolygon(add a polygon to the end of set const GlPolygon& aPolygon); GlStatus InsertPolygon(insert a polygon at a specified const GlPolygon& aPolygon, position int index); GlStatus RemovePolygon(remove a point at a specified int index); position const GlPolygon* GetPolygon(get the polygon at a specified int index) const; position virtual void saveOn(save to a human-readable output ostream &) const;

virtual void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream virtual void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file virtual void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream virtual void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

The following classes are defined in this file: GlPolygonSetP a spcialized parameter for polygon set.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlPolygonP_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlPolygonSetP(); default constructor GlPolygonSetP(const GlPolygonSetP&); copy constructor virtual ~GlPolygonSetP(); destructor

GlPolygonSetP& operator=(assignment operator const GlPolygonSetP&); GlPolygonSetP& operator=(overloaded assignment operator char* templateStr); for converting from SQS string

virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); virtual GlStatus LoadString(load from string stream istrstream& templateStr);

virtual RWCString asString() const; convert to string format virtual RWCString asTemplate() const; convert to SQS format virtual void asString(output in string format ostrstream&) const; to output stream virtual void asTemplate(output in SQS format ostrstream&) const; to output stream

size_t GetNumPolygons() const; get number of polygons GlStatus AddPolygon(add polygon at end const GlPolygon& aPolygon); GlStatus InsertPolygon(insert polygon at specified const GlPolygon& aPolygon, position int index); GlStatus RemovePolygon(int index); remove polygon from specified const GlPolygon* GetPolygon(position int index) const; virtual void saveOn(human-readable output ostream &) const;

void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file void restoreGuts(XDR, etc. input from

RWvistream&); RW-standard stream void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

RWDECLARE_COLLECTABLE(GlPolygonSetP);

RWBoolean deepCompare(compare base value object with const GlParameter&) const; self object GlParameter* NewSpecies() const; create an instance of same class void deepAssign(make a deep copy of self object const GlParameter&);

BEGIN_PROLOG

DESCRIPTION

The following classes are defined in this file: GlPolygon // light weight data structure for a polygon // derived from GlPointSet

The following functions are defined in this file: virtual RWBoolean IsValid() const; // check validity virtual GlStatus AddPoint(// add a point const GlPoint& aPoint); virtual GlStatus InsertPoint(// insert a point const GlPoint& aPoint, int index);

END_PROLOG

Export Control: Public

Inheritance Relationships:

Inherits from GlPointSet

Attributes:

Constructors and Destructor:

Operations:

- AddPoint

  `virtual GlStatus AddPoint(const GlPoint& aPoint);`

  check validity BEGIN_PROLOG

  NAME GlPolygon::AddPoint(const GlPoint& aPoint)

  DESCRIPTION

  This operation adds the input point at the end of the polygon.

  RETURNS GlStatus: GlStatus::SUCCESS if the point is valid and adding it to polygon won't violate validation rule, GlStatus::FAILURE otherwise;

GLOBALS none

FILES none

END_PROLOG

- InsertPoint

```
virtual GlStatus InsertPoint(const GlPoint& aPoint, int index);
```

BEGIN_PROLOG

NAME GlPolygon::InsertPoint(const GlPoint& aPoint, int index)

DESCRIPTION

This operation inserts point at specified location in polygon.

RETURNS GlStatus: GlStatus::SUCCESS if the point is valid and adding it to polygon won't violate validation rule, GlStatus::FAILURE otherwise;

GLOBALS none

FILES none

END_PROLOG

- IsValid

```
virtual int IsValid() const;
```

BEGIN_PROLOG

NAME GlPolygon::IsValid() const

DESCRIPTION

This operation checks the validity of the polygon. Validation rule: valid point set plus 2 extra rules: more than 2 points and no repetition for points.

RETURNS RWBoolean TRUE if valid, FALSE if invalid.

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.22 GlRectangleP Class

Overview:

The following classes are defined in this file: GlRectangleP a spcialized parameter for rectangle.

The following const variables are defined in this file: none

The following defines are defined in this file: _GlRectangleP_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlRectangleP(); default constructor GlRectangleP(const GlRectangleP&); copy constructor GlRectangleP(constructor given all coordinates float n, float s, float e, float w, char *name = 0); GlRectangleP(constructor given two corner points const GlPoint& lowerLeft, const GlPoint& upperRight, char *name = 0); virtual ~GlRectangleP(); destructor

GlRectangleP& operator=(assignment operator const GlRectangleP&); GlRectangleP& operator=(overloaded assignment operator char* templateStr); for converting from SQS string virtual GlStatus LoadTemplate(load from SQS stream istrstream& templateStr); virtual GlStatus LoadString(load from string format istrstream& templateStr);

virtual RWCString asString() const; conver to string format virtual RWCString asTemplate() const; convert to SQS format virtual void asString(output in string format ostrstream& os) const; to output stream virtual void asTemplate(output in SQS format ostrstream& os) const; to output stream void saveOn(human-readable output ostream &) const;

void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream void saveGuts(XDR, etc. output to RWFile&) const; RW-standard file void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream void restoreGuts(XDR, etc. input from RWFile&); RW-standard file

float GetNorth() const; return north coordinate float GetSouth() const; return south coordinate float GetEast() const; return east coordinate float GetWest() const; return west coordinate

void SetNorth(float n); reset north coordinate void SetSouth(float s); reset south coordinate void SetEast(float e); reset east coordinate void SetWest(float w); reset west coordinate

GlPoint GetLowerLeft() const; return lower left corner point GlPoint GetUpperRight() const; return upper right corner point

void Set(set all coordinates float n, float s, float e, float w); void SetLowerLeft(set lower left corner GlPoint& lowerLeft); void SetUpperRight(set upper right corner GlPoint& upperRight);

RWBoolean deepCompare(compare base value object with const GlParameter&) const; self GlParameter* NewSpecies() const; create an instance of same class void deepAssign(make a deep copy of self object const GlParameter&);

Export Control: Public

Inheritance Relationships:

Inherits from GlSpatialP

Attributes:

`myLowerLeft: GlPoint`

`myUpperRight: GlPoint`

lower left point

Constructors and Destructor:

`GlRectangleP(const GlPoint& lowerLeft, const GlPoint& upperRight, char* name);`

`GlRectangleP(const GlRectangleP& rhs);`

default constructor

`GlRectangleP(float n, float s, float e, float w, char* name);`

copy constructor

`GlRectangleP();`

BEGIN_PROLOG

NAME GlRectangleP()

DESCRIPTION

default constructor, required to have this a RWCollectable

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`virtual  ~GlRectangleP();`

BEGIN_PROLOG

NAME ~GlRectangleP()

DESCRIPTION

Destructor.

RETURNS nothing

GLOBALS none

FILES none

Operations:

- asString

  ```
  virtual void asString(ostrstream& os) const;
  ```

  convert to SQS format BEGIN_PROLOG

  NAME asString(ostrstream& os) const

  DESCRIPTION

  This operation writes the value of this object in string format to the output stream. The output format is "(X1, Y1, X2, Y2)".

  RETURNS RWCString.

  GLOBALS none

  FILES none

  END_PROLOG

- asString

  ```
  virtual RWCString asString() const;
  ```

  BEGIN_PROLOG

  NAME asString() const

  DESCRIPTION

  This operation returns the value of this object in string format. The output format is "(X1, Y1, X2, Y2)".

  RETURNS RWCString.

  GLOBALS none

  FILES none

  END_PROLOG

- asTemplate

  ```
  virtual RWCString asTemplate() const;
  ```

  conver to string format BEGIN_PROLOG

  NAME asTemplate() const

  DESCRIPTION

  This operation returns the value of this object in SQS format. The output format is "Rectangle (X1, Y1, X2, Y2)".

  RETURNS RWCString

  GLOBALS none

  FILES none

  END_PROLOG

- asTemplate

  `virtual void asTemplate(ostrstream& os) const;`

  to output stream BEGIN_PROLOG

  NAME asTemplate(ostrstream& os) const

  DESCRIPTION

  This operation writes the value of this object in SQS format to output stream. The output format is "RECTANGLE (X1, Y1, X2, Y2)".

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- copy

  `virtual RWCollectable* copy() const;`

- deepAssign

  `void deepAssign(const GlParameter& rhs);`

  create an instance of same class BEGIN_PROLOG

  NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

  DESCRIPTION

  This operation transfers the inherited and specialized portions of the right-hand-side object to this object, if the objects don't already match.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- deepCompare

  `int deepCompare(const GlParameter& rhs) const;`

  BEGIN_PROLOG

  NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

  DESCRIPTION

  This operation performs a value comparison between the two objects.

  RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

  GLOBALS none

FILES none

END_PROLOG

- GetEast

```
float GetEast() const;
```

return south coordinate BEGIN_PROLOG

NAME GetEast()

DESCRIPTION

This operation returns the east bounding for the rectangle.

RETURNS float

GLOBALS none

FILES none

END_PROLOG

- GetLowerLeft

```
GlPoint GetLowerLeft() const;
```

reset west coordinate BEGIN_PROLOG

NAME GetLowerLeft()

DESCRIPTION

This operation returns the lower left corner point of the rectangle.

RETURNS GlPoint

GLOBALS none

FILES none

END_PROLOG

- GetNorth

```
float GetNorth() const;
```

RW-standard file BEGIN_PROLOG

NAME GetNorth()

DESCRIPTION

This operation returns the north bounding for the rectangle.

RETURNS float

GLOBALS none

FILES none

END_PROLOG

- GetSouth

```
float GetSouth() const;
```

return north coordinate BEGIN_PROLOG

NAME GetSouth()

DESCRIPTION

This operation returns the south bounding for the rectangle.

RETURNS float

GLOBALS none

FILES none

END_PROLOG

- GetUpperRight

```
GlPoint GetUpperRight() const;
```

return lower left corner point BEGIN_PROLOG

NAME GetUpperRight()

DESCRIPTION

This operation returns the upper right corner point of the rectangle.

RETURNS GlPoint

GLOBALS none

FILES none

END_PROLOG

- GetWest

```
float GetWest() const;
```

return east coordinate BEGIN_PROLOG

NAME GetWest()

DESCRIPTION

This operation returns the west bounding for the rectangle.

RETURNS float

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr);
```

BEGIN_PROLOG

NAME LoadString(istrstream& templateStr)

DESCRIPTION

This operation reads the string format from input stream and sets the valus of the objects accordingly. Expected format is "(X1, Y1, X2, Y2)".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE if any conversion failed.

GLOBALS none

FILES none

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

for converting from SQS string BEGIN_PROLOG

NAME LoadTemplate(istrstream& templateStr)

DESCRIPTION

This operation reads the SQS format from input stream and set the valus of the objects accordingly. Expected format is "RECTANGLE (X1, Y1, X2, Y2)".

RETURNS GlSatus, GlStatus::SUCCESS for the correct input and GlStatus::FAILURE for the corrupt input (wrong format).

GLOBALS none

FILES none

END_PROLOG

- NewSpecies

```
GlParameter* NewSpecies() const;
```

self BEGIN_PROLOG

NAME NewSpecies() const

DESCRIPTION

This operation creates a new instance of a GlRectangleP, but returns the pointer to the new object as a (generic) GlParameter*.

RETURNS GlParameter*, the pointer to the newly allocated object

GLOBALS none

FILES none

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- operator =

```
GlRectangleP& operator =(char* templateStr);
```

BEGIN_PROLOG

NAME operator=(char* templateStr)

DESCRIPTION

This operation reads the SQS format from input stream and assign itself with values from the SQS stream. Expected format is "Rectangle (X1, Y1, X2, Y2)".

RETURNS none

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlRectangleP& operator =(const GlRectangleP& rhs);
```

destructor BEGIN_PROLOG

NAME operator=(const const GlRectangleP& other) // right hand side

DESCRIPTION

This operation copies the inherited and specialized information from the right-hand-side object if this object and the other one don't already match.

RETURNS GlCircleP&, reference to self

GLOBALS none

FILES none

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

RW-standard file NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

RW-standard stream BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

RW-standard stream NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

to output stream BEGIN_PROLOG

NAME saveOn(ostream os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a output stream (human readable format).

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetEast

```
void SetEast(float e);
```

reset south coordinate BEGIN_PROLOG

NAME SetEast(float)

DESCRIPTION

This operation resets the east bounding of the rectangle.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetLowerLeft

```
void SetLowerLeft(GlPoint& lowerLeft);
```

BEGIN_PROLOG

NAME SetLowerLeft(GlPoint& lowerLeft)

DESCRIPTION

This operation resets the lower left corner of the rectangle.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- SetNorth

```
void SetNorth(float n);
```

return west coordinate BEGIN_PROLOG

NAME SetNorth(float)

DESCRIPTION

This operation resets the north bounding of the rectangle.
RETURNS void
GLOBALS none
FILES none
END_PROLOG

- SetSouth

```
void SetSouth(float s);
```

reset north coordinate BEGIN_PROLOG
NAME SetSouth(float)
DESCRIPTION
This operation resets the south bounding of the rectangle.
RETURNS void
GLOBALS none
FILES none
END_PROLOG

- SetUpperRight

```
void SetUpperRight(GlPoint& upperRight);
```

BEGIN_PROLOG
NAME SetUpperRight(GlPoint& upperRight)
DESCRIPTION
This operation resets the upper right corner of the rectangle.
RETURNS void
GLOBALS none
FILES none
END_PROLOG

- SetWest

```
void SetWest(float w);
```

reset east coordinate BEGIN_PROLOG
NAME SetWest(float)
DESCRIPTION
This operation resets the west bounding of the rectangle.
RETURNS void
GLOBALS none

FILES none

END_PROLOG

- Set

```
void Set(float n, float s, float e, float w);
```

return upper right corner point BEGIN_PROLOG

NAME Set(float n, float s, float e, float w)

DESCRIPTION

This operation resets all the boundings of the rectangle.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.23 GlSpatialP Class

Overview:

The following classes are defined in this file: GlSpatialP

The following const variables are defined in this file: none

The following defines are defined in this file: _GlSpatialP_h_ make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlSpatialP(); default constructor GlSpatialP(const GlSpatialP&); copy constructor GlSpatialP& operator=(assignment operator const GlSpatial&); virtual ~GlSpatialP(); destructor GlSpatialP(char *name); constructor

virtual RWCString asString() const; return as string virtual RWCString asTemplate() const; return as template

virtual void asString(return as string in output stream ostrstream &) const; virtual void asTemplate(return as template in output stream ostrstream &) const;

virtual GlStatus LoadTemplate(load value from template format istrstream& templateStr); virtual GlStatus LoadString(load value from string format istrstream& templateStr);

virtual RWBoolean IsValid() const; check validity of object

virtual void saveOn(human-readable output ostream &) const; virtual void saveGuts(XDR, etc. output to RWvostream&) const; RW-standard stream virtual void saveGuts(XDR, etc. output to

RWFile&) const; RW-standard file virtual void restoreGuts(XDR, etc. input from RWvistream&); RW-standard stream virtual void restoreGuts(RWFile&); XDR, etc. input from RW-standard file

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

`myValid: int`

destroys heap attrs/composite parameters

Constructors and Destructor:

`GlSpatialP(char* name);`

destructor

`GlSpatialP(const GlSpatialP& rhs);`

default constructor

`GlSpatialP();`

---------- BEGIN_PROLOG

NAME GlSpatialP::GlSpatialP

DESCRIPTION Standard constructor.

RETURNS void

Globals None

Files None

END_PROLOG

`virtual  ~GlSpatialP();`

---------- BEGIN_PROLOG

NAME GlSpatialP::~GlSpatialP

DESCRIPTION Standard destructor, does nothing.

RETURNS void

Globals None

Files None

END_PROLOG

Operations:

- asString

  `virtual void asString(ostrstream& os) const;`

  return as template ----------
  BEGIN_PROLOG

  NAME GlPointP::asString

  DESCRIPTION This method writes the contents of the GlSpatial objects to the passed in stream argument. Derived class should define this in detail.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- asString

  `virtual RWCString asString() const;`

  constructor ---------- BEGIN_PROLOG

  NAME GlPointP::asString

  DESCRIPTION This method returns the contents of the GlSpatial objects in string format. Derived class should define this in detail.

  RETURNS RWString

  Globals None

  Files None

  END_PROLOG

- asTemplate

  `virtual void asTemplate(ostrstream& os) const;`

  ---------- BEGIN_PROLOG

  NAME GlPointP::asTemplate

  DESCRIPTION This method writes the contents of the GlSpatial objects in template format to out stream. Derived class should define this in detail.

  RETURNS RWString

  Globals None

  Files None

  END_PROLOG

- asTemplate

  `virtual RWCString asTemplate() const;`

return as string ---------- BEGIN_PROLOG

NAME GlPointP::asTemplate

DESCRIPTION This method returns the contents of the GlSpatial objects in template format. Derived class should define this in detail.

RETURNS RWString

Globals None

Files None

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
virtual void deepAssign(const GlParameter& rhs);
```

create an instance of same class BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, if the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
virtual int deepCompare(const GlParameter& rhs) const;
```

---------- BEGIN_PROLOG

NAME GlSpatialP::deepCompare(const GlParameter& rhs)

DESCRIPTION Compare too spatial objects, should be defined by derived class.

RETURNS RWBoolean

Globals None

Files None

END_PROLOG

- doRecursiveDelete

```
virtual void doRecursiveDelete(RWTValOrderedVector<class GlParameter*>& );
```

---------- BEGIN_PROLOG

NAME GlSpatialP::doRecursiveDelete()

DESCRIPTION Deleting any recursively defined parameter list. Not applicable for GlSpatialP class.

RETURNS void

Globals None

Files None

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- IsValid

```
virtual int IsValid() const;
```

---------- BEGIN_PROLOG

NAME GlSpatialP::IsValid()

DESCRIPTION Report if the object is valid or not.

RETURNS RWBoolean

Globals None

Files None

END_PROLOG

- LoadString

```
virtual GlStatus LoadString(istrstream& templateStr);
```

---------- BEGIN_PROLOG

NAME GlSpatialP::LoadString(istrstream& templateStr)

DESCRIPTION This is a virtual function, should never be invoked at GlSpatialP level. Derived class should define this.

RETURNS GlStatus

Globals None

Files None

END_PROLOG

- LoadTemplate

```
virtual GlStatus LoadTemplate(istrstream& templateStr);
```

---------- BEGIN_PROLOG

NAME GlSpatialP::LoadTemplate(istrstream& templateStr)

DESCRIPTION This is a virtual function, should never be invoked at GlSpatialP level. Derived class should define this.

RETURNS GlStatus

Globals None

Files None

END_PROLOG

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

```
virtual GlParameter* NewSpecies() const;
```

self object ---------- BEGIN_PROLOG

NAME GlPointP::NewSpecies()

DESCRIPTION This method allocates a new object of type GlSpatialP

RETURNS This method returns a pointer to a GlParameterP containing the address of the new GlSpatialP.

Globals None

Files None

END_PROLOG

- operator =

```
GlSpatialP& operator =(const GlSpatialP& rhs);
```

copy constructor ---------- BEGIN_PROLOG

NAME GlSpatialP::operator=

DESCRIPTION Standard assignment operator.

RETURNS a reference to self after assignment

Globals None

Files None

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

RW-standard stream BEGIN_PROLOG

NAME restoreGuts(RWFile& file) // file from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

RW-standard file NAME restoreGuts(RWvistream& vis) // stream from which to read

DESCRIPTION

This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

RW-standard stream NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

check validity of object BEGIN_PROLOG

NAME saveOn(ostream os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a output stream (human readable format).

RETURNS void

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.24 GlStringP Class

Overview:

The following classes are defined in this file: GlStringP specialization of GlParameter for strings

The following const variables are defined in this file: none

The following defines are defined in this file: GlStringPH make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlStringP(); default constructor needed for RWCollectable GlStringP(const GlStringP&); copy constructor GlStringP(const RWCString& s, normal constructor, parms = const char* name = 0); value and name virtual ~GlStringP(); destructor GlStringP& operator=(assignment operator const GlStringP&); copy from a GlStringP RWCString& operator=(assignment operator const RWCString&); copy from a RWCString const RWCString & value() const; retrieve parameter's value virtual RWCString asString() const; convert to string void saveOn(human-readable output ostream &) const; stream on which to write void saveGuts(XDR, etc. output RWvostream&) const; stream on which to write void saveGuts(

XDR, etc. output RWFile&) const; file to which to write void restoreGuts(XDR, etc. input RWvistream&); stream from which to read void restoreGuts(XDR, etc. input RWFile&); file from which to read RWBoolean deepCompare(compare base value const GlParameter&) const; object with which to compare GlParameter* NewSpecies()const; create new GlStringP pointed to as a GlParameterP void deepAssign(copy including base value const GlParameter&); generic parameter to copy RWspace binaryStoreSize() const; calculate store size including base value size

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

```
myStringBase: RWCString
```

Constructors and Destructor:

```
GlStringP(const RWCString& s, const char* name);
```

copy constructor

```
GlStringP(const GlStringP& rhs);
```

default constructor // needed for RWCollectable

```
GlStringP();
```

```
virtual  ~GlStringP();
```

value and name

Operations:

- asString

```
virtual RWCString asString() const;
```

retrieve parameter's value BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. Since this class actually IS a string, this operation is implemented as a call to the "value()" operation.

RETURNS RWCString, the output from the value() operation.

GLOBALS none

FILES none

END_PROLOG

- binaryStoreSize

```
unsigned long binaryStoreSize() const;
```

generic parameter to copy BEGIN_PROLOG

NAME binaryStoreSize() const

DESCRIPTION

This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

create new GlStringP pointed // to as a GlParameterP BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, IFF the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

required by rogue wave added by jie-bing yu BEGIN_PROLOG

NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

DESCRIPTION

This operation performs a value comparison between the base values of the two objects.

RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

```
GlParameter* NewSpecies() const;
```

object with which to compare BEGIN_PROLOG

NAME NewSpecies() const

DESCRIPTION

This operation creates a new instance of a GlStringP, but returns the pointer to the new object as a (generic) GlParameter*.

RETURNS GlParameter*, the pointer to the newly allocated GlStringP

GLOBALS none

FILES none

END_PROLOG

- operator =

```
GlStringP& operator =(const GlStringP& rhs);
```

destructor BEGIN_PROLOG

NAME operator=(const GlStringP& rhs) // right hand side

DESCRIPTION

This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

RETURNS GlStringP&, reference to self

GLOBALS none

FILES none

END_PROLOG

- operator =

  `RWCString& operator =(const RWCString& s);`

  copy from a GlStringP BEGIN_PROLOG

  NAME operator=(const RWCString& s) // value to store

  DESCRIPTION

  assignment operator, stores the given string as the value of this object

  RETURNS RWCString&, a reference this object's base value

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWvistream& vis);`

  file to which to write BEGIN_PROLOG

  NAME restoreGuts(RWvistream& vis) // stream from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWFile& f);`

  stream from which to read BEGIN_PROLOG

  NAME restoreGuts(RWFile& f) // file from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave file in RW-standard file format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWvostream& vos) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWFile& f) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

convert to string BEGIN_PROLOG

NAME saveOn(ostream &os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a normal stream in a human-readable format

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- value

```
const RWCString& value() const;
```

copy from a RWCString BEGIN_PROLOG

NAME value() const

DESCRIPTION

This operation returns the base value of this object

RETURNS RWCString&, a reference to this object's base value

GLOBALS none

FILES none

END_PROLOG

4.3.18.3.25 GlTimeP Class

Overview:

The following classes are defined in this file: GlTimeP specialization of GlParameter for time data

The following const variables are defined in this file: none

The following defines are defined in this file: GlTimePH make sure this isn't included twice

The following externs are defined in this file: none

The following datatypes are defined in this file: none

The following functions are defined in this file: GlTimeP(); default constructor needed for RWCollectable GlTimeP(const GlTimeP&); copy constructor GlTimeP(const RWTime& d, normal constructor, parms = const char* name = 0); value and name virtual ~GlTimeP(); destructor GlTimeP& operator=(assignment operator const GlTimeP&); copy from a GlTimeP RWTime& operator=(assignment operator const RWTime&); copy from a RWTime const RWTime & value() const; retrieve parameter's value virtual RWCString asString() const; convert to string void saveOn(human-readable output ostream &) const; stream on which to write void saveGuts(XDR, etc. output RWvostream&) const; stream on which to write void saveGuts(XDR, etc. output RWFile&) const; file to which to write void restoreGuts(XDR, etc. input RWvistream&); stream from which to read void restoreGuts(XDR, etc. input RWFile&); file from which to read RWBoolean deepCompare(compare base value const GlParameter&) const; object with which to compare GlParameter* NewSpecies()const; create new GlTimeP pointed to as a GlParameterP void deepAssign(copy including base value const GlParameter&); generic parameter to copy RWspace binaryStoreSize() const; calculate store size including base value size

Export Control: Public

Inheritance Relationships:

Inherits from GlParameter

Attributes:

```
myTimeBase: RWTime
```

Constructors and Destructor:

```
GlTimeP(const RWTime& t, const char* name);
```

copy constructor

```
GlTimeP(const GlTimeP& rhs);
```

default constructor // needed for RWCollectable

```
GlTimeP();
```

```
virtual  ~GlTimeP();
```

value and name

Operations:

- asString

```
virtual RWCString asString() const;
```

retrieve parameter's value BEGIN_PROLOG

NAME asString() const

DESCRIPTION

This operation returns the value of this object in string format. This operation is implemented as a call to the RWTime asString method.

RETURNS RWCString, the output from the value() operation.

GLOBALS none

FILES none

END_PROLOG

- binaryStoreSize

```
unsigned long binaryStoreSize() const;
```

generic parameter to copy BEGIN_PROLOG

NAME binaryStoreSize() const

DESCRIPTION

This operation calculates the amount of space needed to store this object by calculating the space required for the inherited portion and adding to that the space required for this object's value.

RETURNS RWspace, per the rogue wave libraries

GLOBALS none

FILES none

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- deepAssign

```
void deepAssign(const GlParameter& rhs);
```

create new GlTimeP pointed // to as a GlParameterP BEGIN_PROLOG

NAME deepAssign(const GlParameter& rhs) // right hand side of assignment

DESCRIPTION

This operation transfers the inherited and specialized portions of the right-hand-side object to this object, IFF the objects don't already match.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- deepCompare

```
int deepCompare(const GlParameter& rhs) const;
```

required by rogue wave added by jie-bing yu BEGIN_PROLOG

NAME deepCompare(const GlParameter& rhs) const // right hand side of compare

DESCRIPTION

This operation performs a value comparison between the base values of the two objects.

RETURNS RWBoolean, TRUE if the two values match, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- NewSpecies

  ```
  GlParameter* NewSpecies() const;
  ```

  object with which to compare BEGIN_PROLOG

  NAME NewSpecies() const

  DESCRIPTION

  This operation creates a new instance of a GlTimeP, but returns the pointer to the new object as a (generic) GlParameter*.

  RETURNS GlParameter*, the pointer to the newly allocated GlTimeP

  GLOBALS none

  FILES none

  END_PROLOG

- operator =

  ```
  GlTimeP& operator =(const GlTimeP& rhs);
  ```

  destructor BEGIN_PROLOG

  NAME operator=(const GlTimeP& rhs) // right hand side

  DESCRIPTION

  This operation transfers the inherited and specialized information from the right-hand-side object IFF this object and the other one don't already match.

  RETURNS GlTimeP&, reference to self

  GLOBALS none

  FILES none

  END_PROLOG

- operator =

  ```
  RWTime& operator =(const RWTime& t);
  ```

  copy from a GlTimeP BEGIN_PROLOG

  NAME operator=(const RWTime& t) // value to store

  DESCRIPTION

  assignment operator, stores the given date as the value of this object

  RETURNS RWTime&, a reference this object's base value

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWvistream& vis);`

  file to which to write BEGIN_PROLOG

  NAME restoreGuts(RWvistream& vis) // stream from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave virtual input stream in RW-standard input format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- restoreGuts

  `void restoreGuts(RWFile& file);`

  stream from which to read BEGIN_PROLOG

  NAME restoreGuts(RWFile& file) // file from which to read

  DESCRIPTION

  This operation inputs the information for this object from a rogue wave file in RW-standard file format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- saveGuts

  `void saveGuts(RWvostream& vos) const;`

  stream on which to write BEGIN_PROLOG

  NAME saveGuts(RWvostream& vos) const // stream on which to write

  DESCRIPTION

  This operation outputs the information from this object to a rogue wave virtual output stream in RW-standard output format.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- saveGuts

`void saveGuts(RWFile& file) const;`

stream on which to write BEGIN_PROLOG

NAME saveGuts(RWFile& file) const // file to which to write

DESCRIPTION

This operation outputs the information from this object to a rogue wave file in RW-standard file format.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

`void saveOn(ostream& os) const;`

convert to string BEGIN_PROLOG

NAME saveOn(ostream &os) const // stream on which to write

DESCRIPTION

This operation outputs the information from this object to a normal stream in a human-readable format

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- value

`const RWTime& value() const;`

copy from a RWTime BEGIN_PROLOG

NAME value() const

DESCRIPTION

This operation returns the base value of this object

RETURNS RWTime&, a reference to this object's base value

GLOBALS none

FILES none

END_PROLOG

4.3.19 gu Class Category

4.3.19.1 Overview

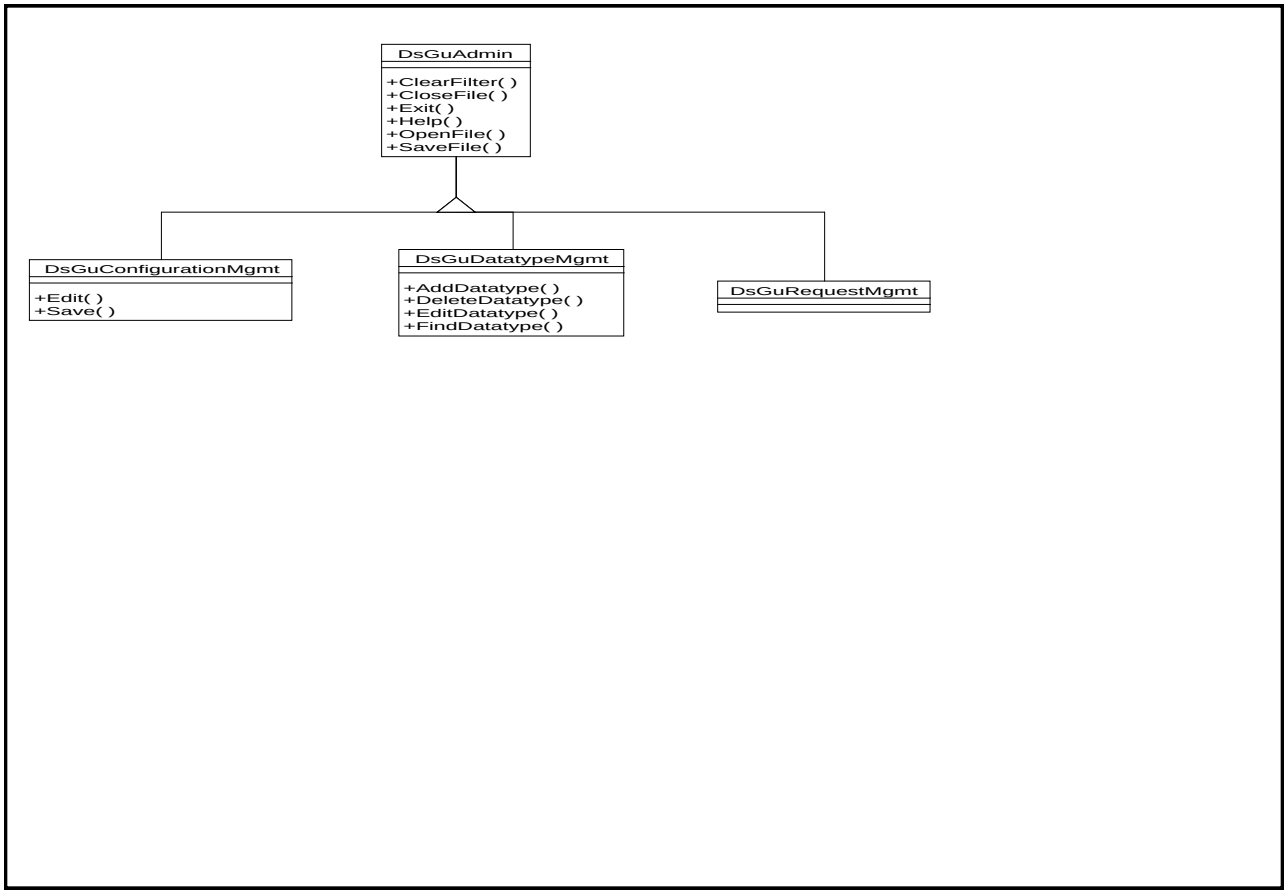


Figure 33 Main

4.3.19.2 gu Classes

4.3.19.2.1 DsGuAdmin Class

Overview:

Distributed Object: No This class is an abstract base class to provide the functions common to all ops/admin screens.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsGuAdmin();
```

The constructor creates an empty object.

```
~DsGuAdmin();
```

Removes an object of this type.

Operations:

- ClearFilter

```
const GlStatus& ClearFilter();
```

This operation unsets any filter selections which may have been set previously.

- CloseFile

```
const GlStatus& CloseFile(fstream, RWCString filename, ifstream,
RWCString filename, ofstream, RWCString filename);
```

This operation closes the provided fstream and assigns the provided name. This operation verifies that the ifstream is the provided file and then closes it. This operation closes the provided ofstream and assigns the provided filename.

- Exit

```
Exit();
```

This function terminates the current screen.

- Help

```
const GlStatus& Help();
```

This function provides context-sensitive help for the current screen.

- OpenFile

```
RWBoolean OpenFile(RWCString filename, fstream&), RWCString
filename, ifstream&, RWCString filename, ofstream&);
```

This service opens the provided file name and assigns it to the provided ifstream. This service opens the provided file name and assigns it to the provided ofstream. This service opens the provided file name and assigns it to the provided fstream.

- SaveFile

```
const GlStatus& SaveFile(RWCString filename);
```

This service saves the current buffer into the specified file and leaves the file open.

- SetAction

```
const GlStatus& SetAction(DsEGuAction);
```

This is the base class function which provides the underlying mechanisms for setting any action button.

- SetFilter

```
const GlStatus& SetFilter(DsEGuFilter);
```

This is the base class function which supplies the basic mechanism for setting any filter.

- SetMode

```
const GlStatus& SetMode(DsEGuMode);
```

This is the base class function which supplies the mechanims for setting any mode.

4.3.19.2.2 DsGuConfigurationMgmt Class

Overview:

Distributed Object: No This class provides the functioning necessary for operators to maintain the various configuration files and/or databases. This class provides the functioning to allow operators to maintain the various configuration files and/or databases.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGuAdmin

Attributes:

Constructors and Destructor:

```
DsGuConfigurationMgmt();
```

The constructor creates an empty object.

```
~DsGuConfigurationMgmt();
```

Removes an object of this type.

Operations:

- Edit

```
Edit();
```

This operation provides the operators the ability to edit (as opposed to just review) the configuration information.

- Save

```
Save();
```

This operation allows the operators to save changes to the configuration information.

4.3.19.2.3 DsGuDatatypeMgmt Class

Overview:

Distributed Object: No This class allows operators to maintain datatypes. This class supplies the functioning to allow operators to maintain datatypes.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGuAdmin

Attributes:

Constructors and Destructor:

```
DsGuDatatypeMgmt();
```

Creates an empty object.

```
~DsGuDatatypeMgmt();
```

Removes an object of this type.

Operations:

- AddDatatype

```
const GlStatus& AddDatatype(CString DLLfileName DsGeTypeID,
RWCString descriptorFile, RWCString DLLfileName);
```

This operation takes the supplied TypeID, descriptor file (name), and DLL filename and adds the specified datatype to the system.

- DeleteDatatype

```
const GlStatus& DeleteDatatype(DsGeTypeID);
```

This operation removes the specified datatype from the system.

- EditDatatype

  ```
  const GlStatus& EditDatatype(DsGeTypeID);
  ```

  This operation allows the operators to edit the DsDeDscriptor for the provided datatype.

- FindDatatype

  ```
  const GlStatus& FindDatatype( version DsEGeTypeCode, RWCString
  name, RWCString version);
  ```

  This operation allows operators to find the DsDescriptor corresponding to the supplied information.

4.3.19.2.4 DsGuRequestMgmt Class

Overview:

Distributed Object: No This class provides the functioning to allow operators to monitor and manage requests.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGuAdmin

Attributes:

Constructors and Destructor:

```
DsGuRequestMgmt();
```

Creates an empty object.

```
~DsGuRequestMgmt();
```

Removes an object of this type.

Operations:

- GetClientInfo

  ```
  MSS_UserProfile GetClientInfo(DsSrRequest&);
  ```

  This operation finds and returns the MSS_UserProfile for the userID specified in the provided request.

- GetRequestHistory

  ```
  istream& history GetRequestHistory(DsSrRequest&);
  ```

This operation returns all the previously recorded state information for the provided request.

- GetRequestInfo

```
istream& information GetRequestInfo(DsSrRequest&);
```

This operation returns information on the current state of the provided request.

- GetRequestResources

```
istream& resourceList GetRequestResources(DsSrRequest&);
```

This operation returns information on the current resource usage of the provided request.

- SetFilterClient

```
RWBoolean SetFilterClient(MSS_UserProfile);
```

This operation allows the operator to select all requests for the provided client.

- SetFilterCSCI

```
RWBoolean SetFilterCSCI(RWCString CSCI);
```

This operation allows the operator to select all requests for the provided CSCI.

- SetFilterPriority

```
RWBoolean SetFilterPriority(RWCString priority);
```

This operation allows the operators to select all requests of the given priority.

- SetFilterProcess

```
RWBoolean SetFilterProcess(RWCString pid);
```

This operation allows the operators to select all the requests for the provided process ID.

- SetFilterState

```
RWBoolean SetFilterState(RWCString state);
```

This operation allows the operators to select all the requests for the provided state (completed, active, queued).

4.3.20 gv Class Category

4.3.20.1 Overview

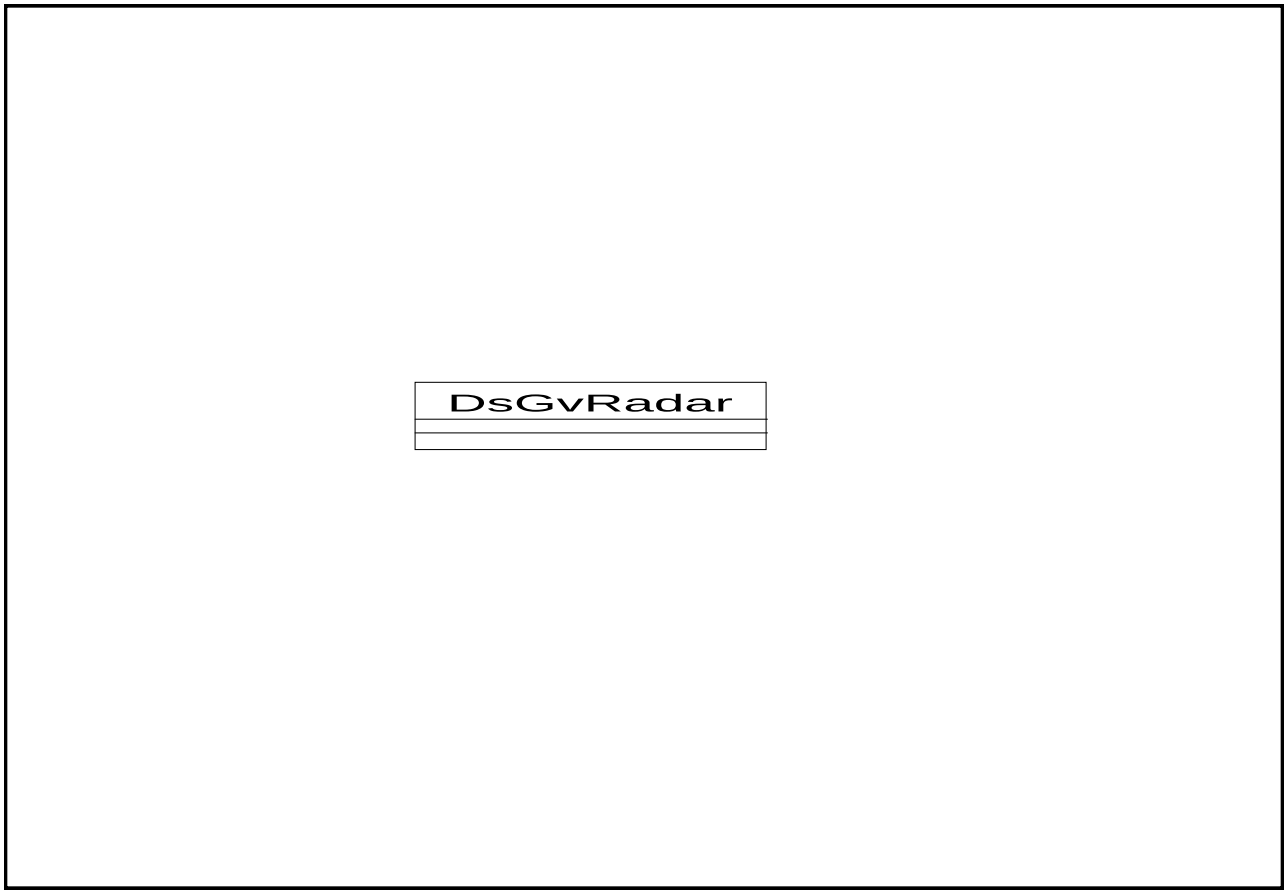


Figure 34 Main

4.3.20.2 gv Classes

4.3.20.2.1 DsGvRadar Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the TRMM Ground Based Validation Radar (GV) network. This network is supported by a network of rain gauges.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

```
DsGvRadar();

~DsGvRadar();
```

Operations:

4.3.21 je Class Category

4.3.21.1 Overview

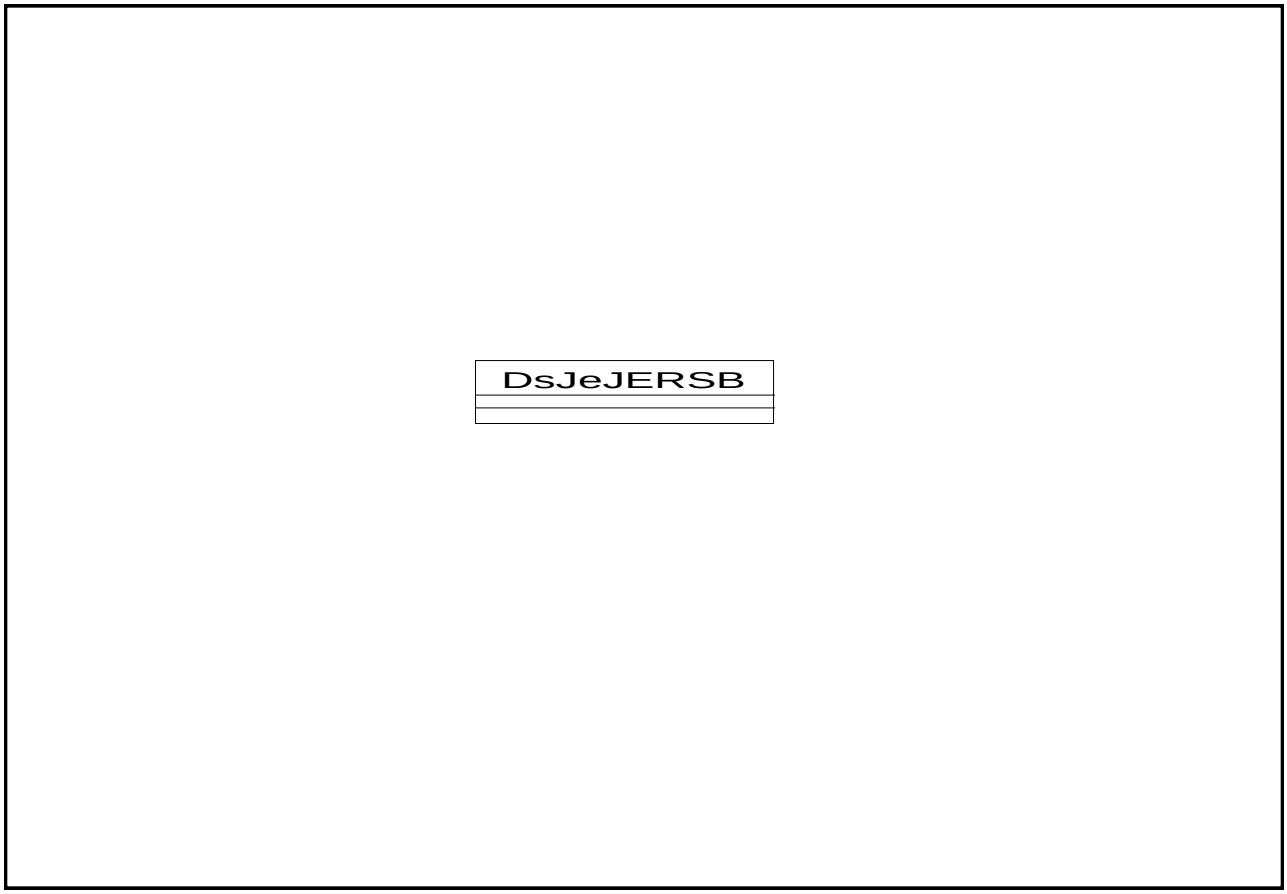


Figure 35 Main

4.3.21.2 je Classes

4.3.21.2.1 DsJeJERSB Class

Overview:

Diustributed Object: No This class is an abstraction of the products derived from the instruments on the JERS satellite.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
~DsJeERSB();
```

Operations:

- DsJeERSB

```
DsJeERSB();
```

4.3.22 md Class Category

4.3.22.1 Overview

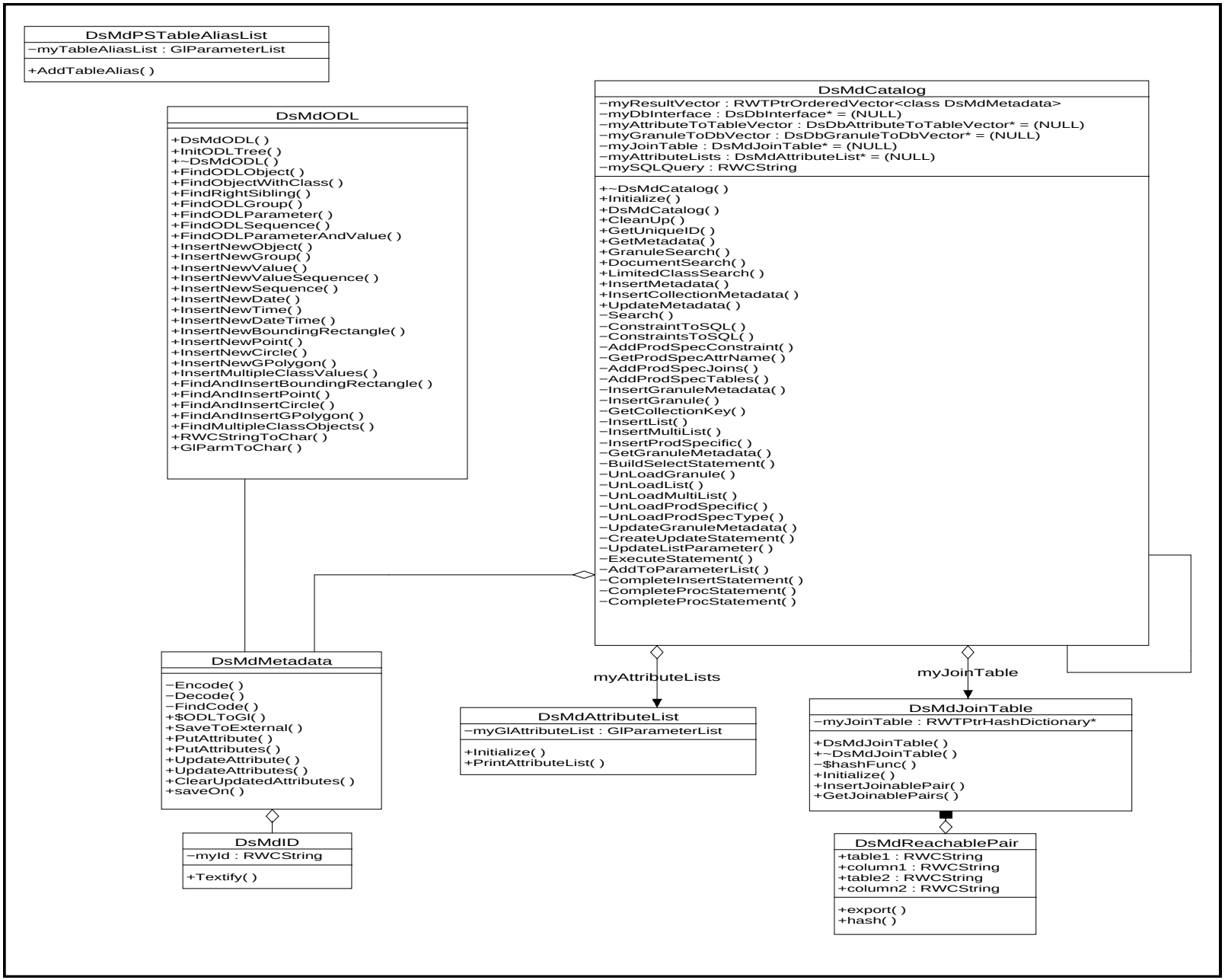


Figure 30 Main

4.3.22.2 md Classes

4.3.22.2.1 DsMdAttributeList Class

Overview:

This file contains the common methods and definitions for the DsMdAttributeList class.

Distributed Object: No Index position of attribute name in one row of data. Md AttributeList Class

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myGlAttributeList: GlParameterList
```

Constructors and Destructor:

```
DsMdAttributeList(void );
```

Constructor BEGIN_PROLOG
NAME DsMdAttributeList
DESCRIPTION
This is the constructor method.
RETURNS None.
GLOBALS None.
FILES none.
END_PROLOG Constructor

```
~DsMdAttributeList(void );
```

Destructor BEGIN_PROLOG
NAME ~DsMdAttributeList
DESCRIPTION
This is the destructor method.
RETURNS None.
GLOBALS None.
FILES none.
END_PROLOG Destructor

Operations:

- GetAttributeList

  ```
  GlParameterList* GetAttributeList(const RWCString& listName);
  ```

  Find and get attributes for given list name. BEGIN_PROLOG

  NAME GetAttributeList

  DESCRIPTION

  This method searches the in-memory attribute list table and returns a pointer to the first entry for the attribute list requested. A NULL pointer is returned if listName is not found.

  RETURNS GlParamterList* Pointer to the GlParamterList containing the attributes requested. A NULL pointer is returned if listName is not found.

  GLOBALS None.

  FILES none.

  END_PROLOG

- Initialize

  ```
  GlStatus Initialize(DsDbInterface* connection);
  ```

  Initialize and load attribute list into memory. BEGIN_PROLOG

  NAME Initialize

  DESCRIPTION

  This method builds the in-memory attribute list from the database.

  RETURNS GlStatus GlStatus object contains either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  GLOBALS None.

  FILES Errors written to cout.

  END_PROLOG

- PrintAttributeList

  ```
  void PrintAttributeList(void );
  ```

  Print the in-memory Attribute List. BEGIN_PROLOG

  NAME PrintAttributeList

  DESCRIPTION

  This method prints the in-memory attribute list to the output stream cout.

  RETURNS None.

  GLOBALS None.

  FILES GlParamterList written to cout.

  END_PROLOG Print the in-memory GlParamerList.

4.3.22.2.2 DsMdCatalog Class

Overview:

This file contains the definitions the DsMdCatalog class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myAttributeLists: DsMdAttributeList*`
used to determine join conditions

`myAttributeToTableVector: DsDbAttributeToTableVector*`
used to access the database

`myDbInterface: DsDbInterface*`
holds the query to execute

`myGranuleToDbVector: DsDbGranuleToDbVector*`
lookup vector for atttribute names

`myJoinTable: DsMdJoinTable*`
lookup vector for determining db of a granule or search condition

`myResultVector: RWTPtrOrderedVector<class DsMdMetadata>`

`mySQLQuery: RWCString`
holds search results

Constructors and Destructor:

`DsMdCatalog();`

`~DsMdCatalog();`
---------- BEGIN_PROLOG
NAME DsMdCatalog::~DsMdCatalog

DESCRIPTION This method is the destructor for the class. It cleans up the member attributes.

RETURNS Void

Globals None

Files None

END_PROLOG

Operations:

- AddProdSpecConstraint

  `GlStatus AddProdSpecConstraint(const GlParameter* singleton, RWCString& condition, DsMdPSTableAliasList& psTableAlias);`

  ---------- BEGIN_PROLOG

  NAME DsMdCatalog::AddProdSpecConstraint

  DESCRIPTION This method adds 2 constraints for a single Product Specific attribute. The constraints are of the form: (ParameterName = "some name" AND ParameterValue = "some value"). The name of the ParameterValue component depends on the type of the Product Specific attribute. This name is retrieved using the GetProdSpecAttrName method. It may be necessary to enhance this method to be smarter about setting the value component of the constraint. The method currently requires that the Gl type of the singleton matches the underlying database type and thus GlParmToAscii will apply quotes to the value if necessary. This method also updates the tableNames hash set with the name of the table containing the product specific attribute.

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- AddProdSpecJoins

  `GlStatus AddProdSpecJoins(RWCString& SQLJoins, const RWCString& defaultTable, int& joinFlag, const DsMdPSTableAliasList& psTableAlias);`

  ---------- BEGIN_PROLOG

  NAME DsMdCatalog::AddProdSpecJoins

  DESCRIPTION This method updates the SQLJoins string with the necessary statements for the product specific attributes. Note a join is required for each product specific attribute in the query.

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

END_PROLOG

- AddProdSpecTables

```
GlStatus AddProdSpecTables(RWCString& SQLQuery, const
DsMdPSTableAliasList& psTableAlias);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::AddProdSpecTables

DESCRIPTION This method updates the SQL from clause string with the table names and alias' necessary for the product specific attributes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- AddToParameterList

```
GlStatus AddToParameterList(GlParameter* pAttributeName,
GlParameterList& theUnloadList);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::AddToParameterList

DESCRIPTION This method creates a GlParameter of the desired type and adds it to the supplied list. The Type is a literal in the GlParameter Description attribute. The Value contains the Name of the new parameter. The argument pAttributeName is required to be a GlStringP.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- BuildSelectStatement

```
GlStatus BuildSelectStatement(GlParameterList& attributeNames,
RWCString& buffer, RWCString id);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::BuildSelectStatement()

DESCRIPTION This method retrieves the attributes from the attributeNames List and builds a SQL select statement.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- CleanUp

```
GlStatus CleanUp();
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::CleanUp

DESCRIPTION This method establishes executes a stored procedure that performs any required clean up activities to support prototypes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- CompleteInsertStatement

```
GlStatus CompleteInsertStatement(DsMdMetadata& metadata,
GlParameterList& insertList, RWCString& buffer, unsigned int
startIndex);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::CompleteInsertStatement

DESCRIPTION This method retrieves each of the supplied attributes (insertList) from the supplied metadata object (metadata) and concatenates their values onto a comma delimited string (buffer) for processing by the database. Attributes processed are at the highest level ParameterList within the metadata object. Convert SQL function is used for Id attributes. The processing of the insertList begins at the position indicated by startIndex. Attributes prior to that have already been processed or are to be skipped.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- CompleteProcStatement

```
GlStatus CompleteProcStatement(DsMdMetadata& metadata,
GlParameterList& insertList, RWCString& buffer, unsigned int
startIndex);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::CompleteProcStatement

DESCRIPTION This method retrieves each of the supplied attributes (insertList) from the supplied metadata object (metadata) and concatenates their values onto a comma delimited string (buffer) for processing by the database. Attributes processed are at the highest level ParameterList within the metadata object.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- CompleteProcStatement

```
GlStatus CompleteProcStatement(const GlParameterList&
metadataList, GlParameterList& insertList, RWCString& buffer,
unsigned int startIndex);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::CompleteProcStatement

DESCRIPTION This method retrieves each of the supplied attributes (insertList) from the supplied GlParameterList (metadataList) and concatenates their values onto a comma delimited string (buffer) for processing by the database. Attributes processed are at the highest level ParameterList within the metadataList object.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- ConstraintsToSQL

```
GlStatus ConstraintsToSQL(const GlParameterList* constraints,
RWCString& conditionClause, RWTPtrHashSet<class RWCString>&
tableNames, DsMdPSTableAliasList& psTableAlias);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::ConstraintsToSQL

DESCRIPTION This method simply calls the ConstraintToSQL method starting the parsing with a "AND" condition between the constraints.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- ConstraintToSQL

```
GlStatus ConstraintToSQL(const GlParameter* singleton,
RWCString& condition, RWTPtrHashSet<class RWCString>&
tableNames, DsMdPSTableAliasList& psTableAlias, int level);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::ConstraintToSQL

DESCRIPTION This method converts the GlParameter argument to a SQL where clause constraint. If the argument is a GlParameterList this method is called recursively with the 1st element and again for each additional element of the list. Additionally the tableName agrument is updated with the name of the table for the attribute.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- CreateUpdateStatement

```
void CreateUpdateStatement(RWCString& buffer, GlParameterList&
pl, const GlParameter* pGranuleId, const RWCString& granIdStr);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::CreateUpdateStatement

DESCRIPTION This method uses the GlParameterList pl to construct a Sybase update statement. The table name is the GlParameterList name, each element of the list contains the column name and new value in string format.

RETURNS This method doesn't return a value. The only error condition is memory errors from the RWCString class.

Globals None

Files None

END_PROLOG

- DocumentSearch

```
GlStatus DocumentSearch(const GlParameterList*
searchConstraints, RWTPtrOrderedVector<class DsMdMetadata>&
results);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::DocumentSearch

DESCRIPTION This method is the entry point for a DsMdCatalog document search. Given a constraint in the form of GlParameterList, this method calls generic Search() with the correct type and StartTable values to control the Search(). This method is used by Document DataServer only.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- ExecuteStatement

```
GlStatus ExecuteStatement(const RWCString& buffer);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::ExecuteStatement

DESCRIPTION This method takes a string buffer containing a database command and executes it.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetCollectionKey

```
GlStatus GetCollectionKey(DsMdMetadata& metadata, RWCString&
collectionKey);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::GetCollectionKey

DESCRIPTION This method retrieves the collection key for the granule using the granule's type and subType attributes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetGranuleMetadata

```
GlStatus GetGranuleMetadata(RWCString id, DsMdMetadata&
metadata, char* (*) [ ] listNames);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::GetGranuleMetadata

DESCRIPTION This method retrieves the granule level metadata asssociated with the DsTMdIdentifier argument id.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetMetadata

```
GlStatus GetMetadata(DsMdMetadata& metadata);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::GetMetadata

DESCRIPTION This method retrieves the metadata asssociated with the DsTMdIdentifier argument id.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetProdSpecAttrName

```
GlStatus GetProdSpecAttrName(const GlParameter* singleton,
RWCString& attrName);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::GetProdSpecAttrName

DESCRIPTION This method determines the name of the attribute that stores the value associated with the given Product Specific attribute constraint. The type of the attribute is retrieved from the database and then used to set the appropriate attribute name.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetUniqueID

```
GlStatus GetUniqueID(RWCString& id);
```

this method removes hard-coded insert granules ---------- BEGIN_PROLOG

NAME DsMdCatalog::GetUniqueID

DESCRIPTION This method retrieves a unique identifier from the Database.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GranuleSearch

```
GlStatus GranuleSearch(const GlParameterList* searchConstraints,
RWTPtrOrderedVector<class DsMdMetadata>& results);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::GranuleSearch

DESCRIPTION This method is the entry point for a DsMdCatalog granule search. Given a constraint in the form of GlParameterList, this method calls generic Search() with the correct type and StartTable values to control the Search().

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- Initialize

```
GlStatus Initialize();
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::Initialize

DESCRIPTION This method establishes a connection to the database through the DsDbInterface. It then initializes it's member attributes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertCollectionMetadata

```
GlStatus InsertCollectionMetadata(DsMdMetadata& metadata);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::InsertCollectionMetadata

DESCRIPTION This method takes a DsMdMetadata argument containing metadata describ, determines it's type, and inserts it's associated metadata attributes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertGranuleMetadata

`GlStatus InsertGranuleMetadata(DsMdMetadata& metadata, const RWCString& granuleIdStr, char* (*) [ ] listNames);`

GlStatus AddBoundingRectConstraints(const GlParameter* singleton, RWCString& condition, RWTPtrHashSet<RWCString>& tableNames); ---------- BEGIN_PROLOG

NAME DsMdCatalog::InsertGranuleMetadata

DESCRIPTION This method takes a DsMdMetadata argument and inserts the associated metadata attributes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertGranule

`GlStatus InsertGranule(DsMdMetadata& metadata, GlParameterList& insertList, const GlParameter* pGranuleId, const char* listType);`

---------- BEGIN_PROLOG

NAME DsMdCatalog::InsertGranule

DESCRIPTION This method inserts the attributes associated with the Granule- InsertList into the database. These atomic attributes are contained in the 1st level GlParameterList of the metadata object.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertList

```
GlStatus InsertList(const GlParameterList& metadataList,
GlParameterList& insertList, const GlParameter* pGranuleId,
const char* listType);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::InsertList

DESCRIPTION This method inserts the attributes associated with the Granule- InsertList into the database. These attributes are contained in the secondary level GlParameterLists within the metadata object.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertMetadata

```
GlStatus InsertMetadata(DsMdMetadata& metadata);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::InsertMetadata

DESCRIPTION This method takes a DsMdMetadata argument, determines it's type, and inserts it's associated metadata attributes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertMultiList

```
GlStatus InsertMultiList(const GlParameterList& metadataList,
GlParameterList& insertList, const GlParameter* pGranuleId);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::InsertMultiList

DESCRIPTION This method inserts the multi-valued attributes in the given metadataList argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertProdSpecific

```
GlStatus InsertProdSpecific(const GlParameterList& metadataList,
const GlParameter* pGranuleId);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::InsertProdSpecific

DESCRIPTION This method inserts the product specific attributes associated with the Granule InsertList into the database. These attributes can be of multiple Gl types and are stored in different tables based upon type. Thus a different stored procedure is executed for each unique type.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- LimitedClassSearch

```
GlStatus LimitedClassSearch(const GlParameterList*
searchConstraints, RWTPtrOrderedVector<class DsMdMetadata>& );
```

- Search

```
GlStatus Search(const GlParameterList* searchConstraints,
RWTPtrOrderedVector<class DsMdMetadata>& results, char*
startTableName, const enum DsGeOID::_basetypetag baseType);
```

holds attributes for insert and retrieve ---------- BEGIN_PROLOG

NAME DsMdCatalog::Search

DESCRIPTION Given a constraint in the form of GlParameterList, this method converts it to a SQL query and executes it through the database wrapper class DsDbInterface, and finally retrieves the resulting granule metadata objects.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- UnLoadGranule

  `GlStatus UnLoadGranule(DsMdMetadata& metadata, GlParameterList& attributeNames);`

  ---------- BEGIN_PROLOG

  NAME DsMdCatalog::UnLoadGranule

  DESCRIPTION This method retrieves the attributes from the attributeNames List and inserts them into the DsMdMetadata argument.

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- UnLoadList

  `GlStatus UnLoadList(GlParameterList& metadataList, GlParameterList& attributeNames, const char* parameterListName);`

  ---------- BEGIN_PROLOG

  NAME DsMdCatalog::UnLoadList

  DESCRIPTION This method retrieves a list structured group of metadata attributes asssociated with a granule.

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- UnLoadMultiList

  `GlStatus UnLoadMultiList(GlParameterList& metadataList, GlParameterList& attributeNames);`

  ---------- BEGIN_PROLOG

  NAME DsMdCatalog::UnLoadMultiList

  DESCRIPTION This method retrieves a multi-valued attribute into the supplied GlParameterList.

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

END_PROLOG

- UnLoadProdSpecific

```
GlStatus UnLoadProdSpecific(DsMdMetadata& metadata);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::UnLoadProdSpecific

DESCRIPTION This method retrieves all Product Specific attributes of the granule, returned as name/value pairs, and inserts them into the metadata object. All product specific attributes are stored as members of one GlParameterList.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- UnLoadProdSpecType

```
GlStatus UnLoadProdSpecType(GlParameterList& pProdSpecList,
GlParameterList& unloadList);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::UnLoadProdSpecType

DESCRIPTION This method retrieves all the Product Specific attributes of one specific type associated with the granule (for example all string attributes). The attributes are returned as name/value pairs and are accumulated into unloadList argument. The first item in unloadList is the name of the attribute, the second item is the value (with the correct type).

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- UpdateGranuleMetadata

```
GlStatus UpdateGranuleMetadata(DsMdMetadata& metadata, const
RWCString& granIdStr);
```

---------- BEGIN_PROLOG

NAME DsMdCatalog::UpdateGranuleMetadata

DESCRIPTION This method takes a DsMdMetadata argument, and updates the attributes in the database associated with the metadata's updated attributes. The main loop examines each attribute, if it is a list, it is processed immediately, otherwise, the single attribute is added to the list of attributes for the attribute's table. All direct table updates are then processed sequentially.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- UpdateListParameter

`GlStatus UpdateListParameter(GlParameterList& updateParms, const GlParameter* pGranuleId);`

---------- BEGIN_PROLOG

NAME DsMdCatalog::UpdateListParameter

DESCRIPTION This method takes a GlParameterList argument, and removes any database records for the attributes in the list and inserts the values supplied in updateParms.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- UpdateMetadata

`GlStatus UpdateMetadata(DsMdMetadata& metadata);`

---------- BEGIN_PROLOG

NAME DsMdCatalog::UpdateMetadata

DESCRIPTION This method takes a DsMdMetadata argument, determines it's type, and updates the attributes in the database associated with the metadata's updated attributes.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

4.3.22.2.3 DsMdID Class

Overview:

This file contains the definitions for the DsMdID class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myId: RWCString
```

Constructors and Destructor:

```
DsMdID(DsMdID& dbId);

DsMdID(RWCString& id);

DsMdID(void );

 ~DsMdID(void );
```

Operations:

- GetID

```
RWCString GetID(void ) const;
```

- Textify

```
RWCString Textify() const;
```

4.3.22.2.4 DsMdJoinTable Class

Overview:

Distributed Object: No Join table class

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myJoinTable: RWTPtrHashDictionary*
```

Constructors and Destructor:

```
DsMdJoinTable();
```

constructor

```
~DsMdJoinTable();
```

destructor ---------- BEGIN_PROLOG

NAME DsMdJoinTable::~DsMdJoinTable

DESCRIPTION This method deletes the memory associated with the joinTable

RETURNS Void

Globals None

Files None

END_PROLOG

Operations:

- GetJoinablePairs

```
RWTPtrSlist<class DsMdReachablePair>* GetJoinablePairs(const
RWCString& tableName) const;
```

retrieve the joinable pair based on first table name ---------- BEGIN_PROLOG

NAME DsMdJoinTable::GetJoinablePairs

DESCRIPTION This method finds all the joinable entries from tableName

RETURNS DsTMdConnListType = RWTPtrSlist<DsMdReachablePair> or NULL if no joinable entries are found.

Globals None

Files None

END_PROLOG

- hashFunc

```
static unsigned int hashFunc(const RWCString& str);
```

- Initialize

```
GlStatus Initialize();
```

---------- BEGIN_PROLOG

NAME DsMdJoinTable::Initialize

DESCRIPTION This method initializes it's joinTable Hash Dictionary with the statically defined joinablePairs

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertJoinablePair

```
void InsertJoinablePair(const RWCString& table1, const RWCString& column1, const RWCString& table2, const RWCString& column2);
```

insert a pair of joinable columns into the table ---------- BEGIN_PROLOG

NAME DsMdJoinTable::InsertJoinablePair

DESCRIPTION This method registers a pair of joinable columns to the join table: the order of the two columns doesn't matter since two entries will be created inside the table to capture the two-way connectivity.

RETURNS Void

Globals None

Files None

END_PROLOG

4.3.22.2.5 DsMdMetadata Class

Overview:

This file contains the definitions the DsMdMetadata class. This class references to external ODL files. The operation of the methods LoadFormExternal and SaveFromExternal rely on the definition of according to the ODL template file Accompanying the SDPS Toolkit 6

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myId: DsGeOID*

myPersistentState: int

myScienceMetadata: GlParameterList*

myUpdatedAttributes: GlParameterList

typeStrings: char**
```

Constructors and Destructor:

```
DsMdMetadata();
```

Constructor // ----------
BEGIN_PROLOG

NAME DsMdMetadata::DsMdMetadata

DESCRIPTION This method is the constructor for the class. It initializes the member attributes.

INPUTS Void

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG

```
DsMdMetadata(const DsMdMetadata& rhs);
```

Copy Constructor // ----------
BEGIN_PROLOG

NAME DsMdMetadata::DsMdMetadata

DESCRIPTION This method is the copy constructor for the class. It initializes the member attributes.

INPUTS const DsMdMetadata& - the metadata object to be copied

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG

~DsMdMetadata();

Destructor // ----------
BEGIN_PROLOG

NAME DsMdMetadata::~DsMdMetadata

DESCRIPTION This method is the destructor for the class. It cleans up the member attributes.

INPUTS None

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG

Operations:

- ClearUpdatedAttributes

GlStatus ClearUpdatedAttributes();

Clear Updated Attributes ----------
BEGIN_PROLOG

NAME DsMdMetadata::ClearUpdatedAttributes

DESCRIPTION Clear Updated Attributes Clears a list of updated metadata attributes in the metadata object

INPUTS None

OUTPUTS None

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- DecodeScienceMetadata

GlStatus DecodeScienceMetadata();

---------- BEGIN_PROLOG

NAME DsMdMetadata::DecodeScienceMetadata

DESCRIPTION This method replaces all occurances of foreign keys within myScienceMetadata to BaseType:SubType:ID form.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- DecodeUpdatedMetadata

`GlStatus DecodeUpdatedMetadata();`

---------- BEGIN_PROLOG

NAME DsMdMetadata::DecodeUpdatedMetadata

DESCRIPTION This method replaces all occurances of foreign keys within myUpdatedMetadata to BaseType:SubType:ID form.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- Decode

`GlStatus Decode(GlParameterList* pParameterList, const RWCString& targetId, const RWCString& targetCodeStr);`

---------- BEGIN_PROLOG

NAME DsMdMetadata::Decode

DESCRIPTION This method replaces the specified foreign keys within the GlParameterList (pParameterList to the BaseType:SubType:ID form using the supplied targetCodeStr.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- EncodeScienceMetadata

`GlStatus EncodeScienceMetadata();`

O: Output stream Initialize the in-memory hash tables fot Attribute Type and Attribute Content GlStatus DsMdMetadata::InitializeAttrTables(DsDbInterface*); Find a named key in the Attribute type dictionary GlStatus DsMdMetadata::FindAttrType(const RWCString& attrKey, DsDeMdMSAttributeType*& attrValue) const; Find a named key in the Attribute content dictionary GlStatus DsMdMetadata::FindAttrContent(const RWCString& attrkey,

DsMdMSAttributeContent*& attrValue) const; ---------- BEGIN_PROLOG

NAME DsMdMetadata::EncodeScienceMetadata

DESCRIPTION This method takes the (string based) DsGeOID foreign references to objects in myScienceMetadata and replaces them with a code and simple db id.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- EncodeUpdatedMetadata

```
GlStatus EncodeUpdatedMetadata();
```

---------- BEGIN_PROLOG

NAME DsMdMetadata::EncodeUpdatedMetadata

DESCRIPTION This method takes the (string based) DsGeOID foreign references to objects in myUpdatedMetadata and replaces them with a code and simple db id.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- Encode

```
GlStatus Encode(GlParameterList* pParameterList, const
RWCString& targetId, const RWCString& targetCodeStr);
```

---------- BEGIN_PROLOG

NAME DsMdMetadata::Encode

DESCRIPTION This method takes the (string based) DsGeOID foreign references to objects in a GlParameterList (pParameterList) and replaces them with a code and simple db id.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindCode

```
unsigned int FindCode(RWCString& s);
```

---------- BEGIN_PROLOG

NAME DsMdMetadata::FindCode

DESCRIPTION This method searches the array of typeStrings to find the supplied string and returns the index within typeStrings.

RETURNS The index within typeStrings containing the string argument or -1 if the string is not found.

Globals typeString : array of types

Files None

END_PROLOG

- GetAttributes

`GlStatus GetAttributes(GlParameterList& attributes) const;`

Get Metadata Attributes Get a list of attributes from the metadata object ---------- BEGIN_PROLOG

NAME DsMdMetadata::GetAttributes

DESCRIPTION Get Metadata Attributes Get a list of attributes from the metadata object Note: The input attributes should already contain the attribute names of those to be retrieved. The values are filled upon completion. Be aware that the values are shallow copied.

INPUTS GlParameterList& - A GlParameterList containing the required attributes

OUTPUTS None

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetAttribute

`GlStatus GetAttribute(const RWCString& attributeName, const GlParameter*& attribute) const;`

Get Metadata Attribute Gets an attribute from the Metadata object ---------- BEGIN_PROLOG

NAME DsMdMetadata::GetAttribute

DESCRIPTION Gets an attribute from the Metadata object

INPUTS const RWCString& - The attribute name to find

OUTPUTS const GlParameter*& - the found GlParameter containing the attribute value

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetDeepAttribute

```
GlStatus GetDeepAttribute(RWInteger sequenceType, const RWCString& attributeName, const GlParameter*& attribute, const GlParameter*& lastAttributeFound, GlParameterList list) const;
```

Get Metadata Attribute Gets an attribute from the Metadata object ---------- BEGIN_PROLOG

NAME DsMdMetadata::GetDeepAttribute

DESCRIPTION Gets an attribute from a deep attribute list

INPUTS RWInteger - the sequence type of the attribute const RWCString& - The attribute name to find const GlParameter*& - the GlParameter containing the last attribute value found GlParameterList - the GlParameterList containing the attributes or list of attributes

OUTPUTS const GlParameter*& - the found GlParameter containing the attribute value const GlParameter*& - the GlParameter containing the last attribute value found

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetMyId

```
DsGeOID* GetMyId() const;
```

- GetMyPersistentState

```
const int& GetMyPersistentState() const;
```

- GetScienceMetadataList

```
const GlParameterList* GetScienceMetadataList() const;
```

- GetUpdatedAttributes

```
GlStatus GetUpdatedAttributes(GlParameterList& modifiedAttributes) const;
```

Get Updated Attributes ---------- BEGIN_PROLOG

NAME DsMdMetadata::GetUpdatedAttributes

DESCRIPTION Get Updated Attributes Gets a list of updated metadata attributes in the metadata object

INPUTS None

OUTPUTS GlParameterList& - The list of modified attributes

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- ODLToGl

`static GlStatus ODLToGl(istream& externalForm, GlParameterList& aMetadataList);`

Load from External Imports the metadata from its external format to its in-memory format Parses the ODL string and stores the attributes in their native format in the GlParameterList ---------- BEGIN_PROLOG

NAME DsMdMetadata::ODLToGl

DESCRIPTION Imports the metadata from its external format to its in-memory format Parses the ODL string and stores the attributes in their native format in the GlParameterList

INPUTS FILE* - pointer to input file

OUTPUTS None

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- operator ==

`int operator ==(const DsMdMetadata& metadata);`

Overloaded operator== ---------- BEGIN_PROLOG

NAME DsMdMetadata::operator==

DESCRIPTION Overloaded operator==

INPUTS const DsMdMetadata& - the metadata object subject to the equality

OUTPUTS None

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- PutAttributes

`GlStatus PutAttributes(GlParameterList& attributes);`

Put Metadata Attributes Puts a list of attributes to the metadata object ---------- BEGIN_PROLOG

NAME DsMdMetadata::PutAttributes

DESCRIPTION Put Metadata Attributes Puts a list of attributes to the metadata object

INPUTS GlParameterList& - the list of attributes to be put into the metadata object

OUTPUTS None

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- PutAttribute

`GlStatus PutAttribute(GlParameter* attribute);`

Put Metadata Attribute Puts an attribute to the metadata object ---------- BEGIN_PROLOG

NAME DsMdMetadata::PutAttribute

DESCRIPTION Put Metadata Attribute Puts an attribute to the metadata object Note: 1. A copy of the attribute is made and stored upon invocation. 2. Multi-valued attribute will be put into a GlParameterList with the same attribute name. 3. The state variables: myId and myType are automatically updated when 'granuleId' or 'shortName' are put. INPUTS GlParameter* - The attribute to put into the metadata object

OUTPUTS None

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- saveOnUpdated

`void saveOnUpdated(ostream& os) const;`

O: Output stream Save internals of UpdatedAttributes on output stream

- saveOn

  `void saveOn(ostream& os) const;`

  Save internals of myScienceMetadata on output stream

- SaveToExternal

  `GlStatus SaveToExternal(stdio_h_00010* MSTemplate, stdio_h_00010*& externalForm);`

  SaveToExternal Exports the in-memory format of the metadata to its external format Takes a GlParameterList and constructs a ODL string ---------- BEGIN_PROLOG

  NAME DsMdMetadata::SaveToExternal

  DESCRIPTION Exports the in-memory format of the metadata to its external format Takes a GlParameterList and constructs an ODL string

  OUTPUTS char*& - pointer to a file

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- SetMyId

  `void SetMyId(DsGeOID* pId);`

  Set MyId ---------- BEGIN_PROLOG

  NAME DsMdMetadata::SetMyID

  DESCRIPTION This method sets the myId attribute of the metadata object to a copy of DsGeOID passed in. It also inserts the 3 DsGeOID attributes into myScienceMetadata for DB processing.

  INPUTS DsGeOID* pId address

  OUTPUTS None

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- SetMyPersistentState

  `void SetMyPersistentState(const int& state);`

  Set MyPersistentState

- SetParameterList

  `void SetParameterList(GlParameterList* pParameterList);`

  ---------- BEGIN_PROLOG

  NAME DsMdMetadata::SetParameterList

  DESCRIPTION This method sets the myScienceMetadata attribute of the metadata object to the passed in GlParameterList.

  INPUTS GlParameterList* pParameterList

  OUTPUTS None

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- UpdateAttributes

  `GlStatus UpdateAttributes(GlParameterList& attributes);`

  Update Metadata Attributes Updates a list of metadata attributes in the metadata object ---------- BEGIN_PROLOG

  NAME DsMdMetadata::UpdateAttributes

  DESCRIPTION Update Metadata Attributes Updates a list of metadata attributes in the metadata object

  INPUTS GlParameterList& - The list of attributes to be updated in the metadata object

  OUTPUTS None

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- UpdateAttribute

  `GlStatus UpdateAttribute(GlParameter* attribute);`

  Update Metadata Attribute Updates the specified metadata attribute in the metadata object ---------- BEGIN_PROLOG

  NAME DsMdMetadata::UpdateAttribute

  DESCRIPTION Update Metadata Attribute Updates the specified metadata attribute in the metadata object

  INPUTS GlParameter* - the attribute to be updated

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

4.3.22.2.6 DsMdODL Class

Overview:

This file contains the definitions the DsMdODL class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

`DsMdODL();`

Constructor // ----------
BEGIN_PROLOG

NAME DsMdODL::DsMdODL

DESCRIPTION This method is the constructor for the class.

INPUTS Void

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG

`~DsMdODL();`

Copy Constructor // DsMdODL(const DsMdODL&); Destructor // ---------- BEGIN_PROLOG

NAME DsMdODL::DsMdODL

DESCRIPTION This method is the copy constructor for the class.

INPUTS const DsMdODL& - the ODL object to be copied

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG DsMdODL::DsMdODL (const DsMdODL& rhs) { } ---------- BEGIN_PROLOG

NAME DsMdODL::~DsMdODL

DESCRIPTION This method is the destructor for the class.

INPUTS None

OUTPUTS None

RETURNS None

Globals None

Files None

END_PROLOG

Operations:

- FindAndInsertBoundingRectangle

```
GlStatus FindAndInsertBoundingRectangle(Aggregate_Node*
initGroupNode, char* initMSattr, GlRectangleP* rect,
GlParameterList& list, const RWTPtrHashDictionary<class
RWCString, class DsDeAttributeType>* MSAttributeType);
```

Insert a new GlRectangleP into specified GlParameterList ---------- BEGIN_PROLOG

NAME DsMdODL::FindAndInsertBoundingRectangle

DESCRIPTION Find values for and Insert a new GlRectangleP into specified GlParameterList

INPUTS AGGREGATE - the group node for this group char* - the attribute name in the MS GlRectangleP* - the GlRectangle containing the bounding rectangle values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

OUTPUTS GlParameterList*& - the GlParameterList containing the found parameters and their values// RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindAndInsertCircle

  GlStatus FindAndInsertCircle(Aggregate_Node* groupNode, char* initMSattr, GlCircleP* circ, GlParameterList& list, const RWTPtrHashDictionary<class RWCString, class DsDeAttributeType>* MSAttributeType);

  Insert a new GlCircleP into specified GlParameterList ---------- BEGIN_PROLOG

  NAME DsMdODL::FindAndInsertCircle

  DESCRIPTION Find values for and Insert a new GlCircle into specified GlParameterList

  INPUTS AGGREGATE - the group node for this group char* - the attribute name in the MS GlCircleP* - the GlCircleP containing the circle values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

  OUTPUTS GlParameterList*& - the GlParameterList containing the found parameters and their values// RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- FindAndInsertGPolygon

  GlStatus FindAndInsertGPolygon(Aggregate_Node* initGroupNode, GlGPolygonP* gPoly, GlParameterList& list, const RWTPtrHashDictionary<class RWCString, class DsDeAttributeType>* MSAttributeType);

  Insert a new GlGPolygonP into science metadata GlParameterList ---------- BEGIN_PROLOG

  NAME DsMdODL::FindAndInsertGPolygon

  DESCRIPTION Insert a new GlGPolygonP into science metadata GlParameterList

  INPUTS AGGREGATE& - the group node where the values are to be pasted char* - the attribute name in the MS GlGPolygonP* - the GlGPolygon containing the boudary GRing and Exclusion GRing(s) GlParameterList*& - the GlParameterList in which to insert the GlParameterList RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

  OUTPUTS AGGREGATE& - the group node where the values are to be pasted

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- FindAndInsertPoint

  GlStatus FindAndInsertPoint(Aggregate_Node* groupNode, char* initMSattr, GlPointP* pnt, GlParameterList& list, const RWTPtrHashDictionary<class RWCString, class DsDeAttributeType>* MSAttributeType);

  Insert a new GlPoint into specified GlParameterList ---------- BEGIN_PROLOG

  NAME DsMdODL::FindAndInsertPoint

  DESCRIPTION Find values for and Insert a new GlPoint into specified GlParameterList

  INPUTS AGGREGATE - the group node for this group char* - the attribute name in the MS GlPoint* - the GlPoint containing the point values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

  OUTPUTS GlParameterList*& - the GlParameterList containing the found parameters and their values// RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

  Files None

  END_PROLOG

- FindMultipleClassObjects

  GlStatus FindMultipleClassObjects(Aggregate_Node* groupNode, GlParameterList*& list, const RWTPtrHashDictionary<class RWCString, class DsDeAttributeType>* MSAttributeType, const RWTPtrHashDictionary<class RWCString, class DsDeAttributeContent>* MSAttributeContent);

  Finds objects within a Multiple Class Group and inserts these into the specified GlParameterList ---------- BEGIN_PROLOG

  NAME DsMdODL::FindMultipleClassObjects

  DESCRIPTION Finds objects within a Multiple Class Group and inserts these into the specified GlParameterList

  INPUTS AGGREGATE& - the group node containing the values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table RWTPtrHashDictionary<RWCString, DsMdMSAttributeContent>* - the attribute group look-up table

  OUTPUTS GlParameterList*& - the GlParameterList in which to insert the GlParameterList of found values

  RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

  Globals None

Files None

END_PROLOG

- FindObjectWithClass

```
Aggregate_Node* FindObjectWithClass(Aggregate_Node* baseNode,
char* name, char* classification);
```

Find a named object with the specified class, searches from base node and continues as necessary with all the base node's children, all their children, using a preorder search

- FindODLGroup

```
GlStatus FindODLGroup(Aggregate_Node* masterNode, char* MSattr,
char [ ] [ ] mdHandles, int groupCount, char* groupName,
GlParameterList*& list, const RWTPtrHashDictionary<class
RWCString, class DsDeAttributeType>* MSAttributeType);
```

Find a named group in the ODL tree ---------- BEGIN_PROLOG

NAME DsMdODL::FindODLGroup

DESCRIPTION Find a named group in the ODL tree

INPUTS AGGREGATE& - the master node for this MS file char* - the attribute name in the MS allHandles - the array containing the group handles for this MS file int - the group counter char* - the group name RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

OUTPUTS GlParameterList*& - the GlParameter list containing the found parameters and their values

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindODLObject

```
GlStatus FindODLObject(RWCString object, Aggregate_Node*&
groupNode, char* MSattr, Value_Node*& valueNode, char*
classValue);
```

Find a named object in the ODL tree ---------- BEGIN_PROLOG

NAME DsMdODL::FindODLObject

DESCRIPTION Find a named object in the ODL tree

INPUTS RWCString - the string containing the object name to find AGGREGATE& - the master node for this MS file char* - the attribute name in the MS char* - the class value in the MS

OUTPUTS VALUE& - the structure containing the found parameter and its value

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindODLParameterAndValue

`GlStatus FindODLParameterAndValue(Aggregate_Node* objectNode, char* MSattrParam, Parameter_Node*& attrNode, Value_Node*& valueNode);`

Find a named parameter node and value in ther named object ---------- BEGIN_PROLOG

NAME DsMdODL::FindODLParameterAndValue

DESCRIPTION Find a named parameter node and value in ther named object

INPUTS OBJECT - the object node containing the parameter and value char* - the attribute name in the MS char* - the attribute name in the MS PARAMETER& - the parameter node to find

OUTPUTS VALUE& - the structure containing the found value

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindODLParameter

`GlStatus FindODLParameter(RWCString object, Aggregate_Node*& groupNode, char* MSattr, Parameter_Node*& attrNode);`

Find a named ODL parameter in the ODL tree ---------- BEGIN_PROLOG

NAME DsMdODL::FindODLParameter

DESCRIPTION Find a named parameter in the ODL tree, get the parameter and value(s)

INPUTS RWCString - the string containing the object name to find AGGREGATE& - the master node for this MS file char* - the attribute name in the MS

OUTPUTS VALUE& - the structure containing the found parameter and its value

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- FindODLSequence

```
GlStatus FindODLSequence(RWCString object, RWInteger objType,
Aggregate_Node*& groupNode, char* MSattr, GlParameterList*&
sequence, char* classValue);
```

Find a named sequence object in the ODL tree ---------- BEGIN_PROLOG

NAME DsMdODL::FindODLSequence

INPUTS RWCString - the string containing the object name to find RWInteger - the string containing the object type AGGREGATE& - the master node for this MS file char* - the attribute name in the MS char* - the class value in the MS

OUTPUTS GlParameterList*& - the GlParameterList containing the found parameters and its values

DESCRIPTION Find a named sequence object in the ODL tree and insert into a GlParameter List

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG Find a named sequence object in the ODL tree and insert into a GlParameter List

- FindRightSibling

```
Aggregate_Node* FindRightSibling(Aggregate_Node* baseNode);
```

Find the right sibling

- GlParmToChar

```
char* GlParmToChar(const GlParameter* attrParam);
```

Local GlParameter value to char* conversion

- InitODLTree

```
GlStatus InitODLTree(stdio_h_00010* fp, char [ ] [ ] mdHandles,
Aggregate_Node*& masterNode);
```

Initialize ODL tree ---------- BEGIN_PROLOG

NAME DsMdODL::InitODLTree

DESCRIPTION Initialize ODL tree

INPUTS FILE* - the file pointer to the MS file allHandles - the array containing the group handles for this MS file AGGREGATE& - the master node for this MS file

OUTPUTS None

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertMultipleClassValues

```
GlStatus InsertMultipleClassValues(Aggregate_Node*& groupNode,
const GlParameter* attrParam, const RWTPtrHashDictionary<class
RWCString, class DsDeAttributeType>* MSAttributeType, const
RWTPtrHashDictionary<class RWCString, class
DsDeAttributeContent>* MSAttributeContent);
```

Insert a new GlParameterList containing multiple values into specified Group node ---------- BEGIN_PROLOG

NAME DsMdODL::InsertMultipleClassValues

DESCRIPTION Insert a new GlParameterList containing multiple values into specified Group node

INPUTS AGGREGATE& - the group node where the found values are to be pasted const GlParameter* - the GlParameter pointing to the values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table RWTPtrHashDictionary<RWCString, DsMdMSAttributeContent>* - the attribute group look-up table OUTPUTS AGGREGATE& - the group node where the found values are to be pasted

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewBoundingRectangle

```
GlStatus InsertNewBoundingRectangle(Aggregate_Node*& groupNode,
char* initMSattr, GlRectangleP* rect, const
RWTPtrHashDictionary<class RWCString, class DsDeAttributeType>*
MSAttributeType);
```

Insert a new GlRectangleP into specified Group node ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewBoundingRectangle

DESCRIPTION Insert a new GlRectangleP into specified Group node

INPUTS AGGREGATE& - the group node where the values are to be pasted char* - the attribute name in the MS GlRectangleP* - the GlRectangleP containing the found parameters and their values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

OUTPUTS AGGREGATE& - the group node where the values are to be pasted

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewCircle

```
GlStatus InsertNewCircle(Aggregate_Node*& groupNode, char*
initMSattr, GlCircleP* circ, const RWTPtrHashDictionary<class
RWCString, class DsDeAttributeType>* MSAttributeType);
```

Insert a new GlCircleP into specified Group node ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewCircle

DESCRIPTION Insert a new GlCircleP into specified Group node

INPUTS AGGREGATE& - the group node where the values are to be pasted char* - the attribute name in the MS GlCircleP* - the GlPoint containing the found parameters and their values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

OUTPUTS AGGREGATE& - the group node where the values are to be pasted

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewDateTime

```
GlStatus InsertNewDateTime(Value_Node*& valueNode,
GlParameterList& list);
```

Insert a new datetime into specified GlParameterList

- InsertNewDate

```
GlStatus InsertNewDate(Value_Node*& valueNode, GlParameterList&
list);
```

Insert a new GlDate into specified GlParameterList

- InsertNewGPolygon

GlStatus InsertNewGPolygon(Aggregate_Node*& groupNode, GlGPolygonP* gPoly, const RWTPtrHashDictionary<class RWCString, class DsDeAttributeType>* MSAttributeType);

Insert a new GlGPolygonP into specified Group node ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewGPolygon

DESCRIPTION Insert a new GlGPolygonP into specified Group node

INPUTS AGGREGATE& - The group node containing the candidate GRing structure GlGPolygonP* - the GlGPolygon containing the boudary GRing and Exclusion GRing(s) RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

OUTPUTS AGGREGATE& - The group node containing the candidate GRing structure

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewGroup

GlStatus InsertNewGroup(GlParameterList* myGroup, GlParameterList*& list);

Insert a new GlParameterList containing group of objects into specified GlParameterList ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewGroup

DESCRIPTION Insert a new GlParameterList containing group of objects into specified GlParameterList

INPUTS GlParameterList*& - the GlParameterList containing the found parameters and their values

OUTPUTS GlParameterList*& - the GlParameterList in which to insert the GlParameterList

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewObject

GlStatus InsertNewObject(Value_Node*& valueNode, RWCString object, RWInteger objType, GlParameterList*& list);

Insert a new GlParameter containing one object into specified GlParameterList ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewObject

DESCRIPTION Insert a new GlParameter containing a value into specified GlParameterList

INPUTS VALUE& - the structure containing the parameters and values for this object RWCString - the string containing the object name to insert RWCString - the string containing the object type GlParameterList*& - the GlParameterList containing the found parameters and their values

OUTPUTS GlParameterList*& - the GlParameterList containing the found parameters and their values

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewPoint

```
GlStatus InsertNewPoint(Aggregate_Node*& groupNode, char*
initMSattr, GlPointP* pnt, const RWTPtrHashDictionary<class
RWCString, class DsDeAttributeType>* MSAttributeType);
```

Insert a new GlPoint into specified Group node ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewPoint

DESCRIPTION Insert a new GlPoint into specified Group node

INPUTS AGGREGATE& - the group node where the values are to be pasted char* - the attribute name in the MS GlPoint* - the GlPoint containing the found parameters and their values RWTPtrHashDictionary<RWCString, DsMdMSAttributeType>* - the attribute look-up table

OUTPUTS AGGREGATE& - the group node where the values are to be pasted

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewSequence

```
GlStatus InsertNewSequence(GlParameterList* sequence,
GlParameterList*& list);
```

Insert a new GlParameterList containing a sequence into specified GlParameterList ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewSequence

DESCRIPTION Insert a new GlParameterList containing a sequence into specified GlParameterList

INPUTS GlParameterList*& - the GlParameterList containing the found sequence of parameters and their values

OUTPUTS GlParameterList*& - the GlParameterList in which to insert the GlParameterList RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewTime

```
GlStatus InsertNewTime(Value_Node*& valueNode, GlParameterList& list);
```

Insert a new GlTime into specified GlParameterList

- InsertNewValueSequence

```
GlStatus InsertNewValueSequence(RWInteger attributeType, Parameter_Node* attrNode, const GlParameter*& attrParam);
```

Insert a new sequence of values obtained from a GlParameterList into the specified ValueNode ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewValueSequence

DESCRIPTION Insert a new sequence of values obtained from a GlParameterList into the specified ValueNode

INPUTS RWInteger - the attribute type of the object PARAMETER - the parameter node wher the values are to be pasted const GlParameterList*& - the GlParameterList containing the found sequence of parameters and their values

OUTPUTS const GlParameterList*& - the GlParameterList containing the found sequence of parameters and their values

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- InsertNewValue

```
GlStatus InsertNewValue(RWInteger attributeType, Parameter_Node* attrNode, const GlParameter*& attrParam);
```

Insert a new value obtained from a GlParameterList into the specified ValueNode ---------- BEGIN_PROLOG

NAME DsMdODL::InsertNewValue

DESCRIPTION Insert a new value obtained from a GlParameterList into the specified ValueNode

INPUTS RWInteger - the attribute type of the object PARAMETER - the parameter node where the values are to be pasted const GlParameter*& - the GlParameter containing the found values

OUTPUTS const GlParameter*& - the GlParameter containing the found values

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- RWCStringToChar

```
char* RWCStringToChar(RWCString myRWCString);
```

Local RWCString to char* conversion

4.3.22.2.7 DsMdPSTableAliasList Class

Overview:

This file contains the definition of the DsMdPSTableAliasList class.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myTableAliasList: GlParameterList
```

Constructors and Destructor:

```
DsMdPSTableAliasList(void );
```

---------- BEGIN_PROLOG

NAME DsMdPSTableAliasList::DsMdPSTableAliasList

DESCRIPTION Constructor, does nothing.

RETURNS void

Globals None

Files None

END_PROLOG

~DsMdPSTableAliasList(void);

---------- BEGIN_PROLOG

NAME DsMdPSTableAliasList::~DsMdPSTableAliasList

DESCRIPTION Destructor, removes elements from the list of product specific table names and alias'.

RETURNS void

Globals None

Files None

END_PROLOG

Operations:

- AddTableAlias

  void AddTableAlias(const RWCString& tableName, RWCString& alias);

  ---------- BEGIN_PROLOG

  NAME DsMdPSTableAliasList:: AddTableAlias

  DESCRIPTION This method inserts a Product Specific table name and the alias to be used for this search into it's list and returns the selected alias in the alias argument.

  RETURNS void

  Globals None

  Files None

  END_PROLOG

- GetNumTableAlias

  unsigned int GetNumTableAlias() const;

  ---------- BEGIN_PROLOG

  NAME DsMdPSTableAliasList::GetNumTableAlias

  DESCRIPTION This method inserts the multi-valued attributes in the given metadataList argument.

  RETURNS size_t number of elements in myTableAliasList.

  Globals None

Files None

END_PROLOG

- GetTableAlias

```
GlStatus GetTableAlias(RWCString& alias, unsigned int index)
const;
```

---------- BEGIN_PROLOG

NAME DsMdPSTableAliasList::GetTableAlias

DESCRIPTION This method retrieves the table alias at the given location within myTableAliasList and stores it in the alias argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

- GetTableName

```
GlStatus GetTableName(RWCString& tableName, unsigned int index)
const;
```

---------- BEGIN_PROLOG

NAME DsMdPSTableAliasList::GetTableName

DESCRIPTION This method retrieves the table name at the given location within myTableAliasList into the tableName argument.

RETURNS This method returns a GlStatus object. This GlStatus will contain either GlStatus::SUCCESS or GlStatus::FAILURE, and a brief error message.

Globals None

Files None

END_PROLOG

4.3.22.2.8 DsMdReachablePair Class

Overview:

Distributed Object: No #include <DsMdTypes.h>

Export Control: Implementation

Inheritance Relationships:

Attributes:

`column1: RWCString`

table1 name

`column2: RWCString`

table2 name

`table1: RWCString`

table1.column1 can be joined with table2.column2

`table2: RWCString`

table1 column

Constructors and Destructor:

```
DsMdReachablePair(const RWCString& tab1, const RWCString& col1,
const RWCString& tab2, const RWCString& col2);
```

table2 column constructor

Operations:

- export

```
void export(const DsDbAttributeToTableVector* map, RWCString&
obuf);
```

export the joinable pair to acsii format for SQL statement

- GetOther

```
GlStatus GetOther(const RWCString& table, RWCString& other);
```

given a table name, return the other table name appeared in the pair

- hash

```
unsigned int hash() const;
```

hash function needed for search

- operator ==

```
int operator ==(const DsMdReachablePair& other) const;
```

4.3.23 mo Class Category

4.3.23.1 Overview

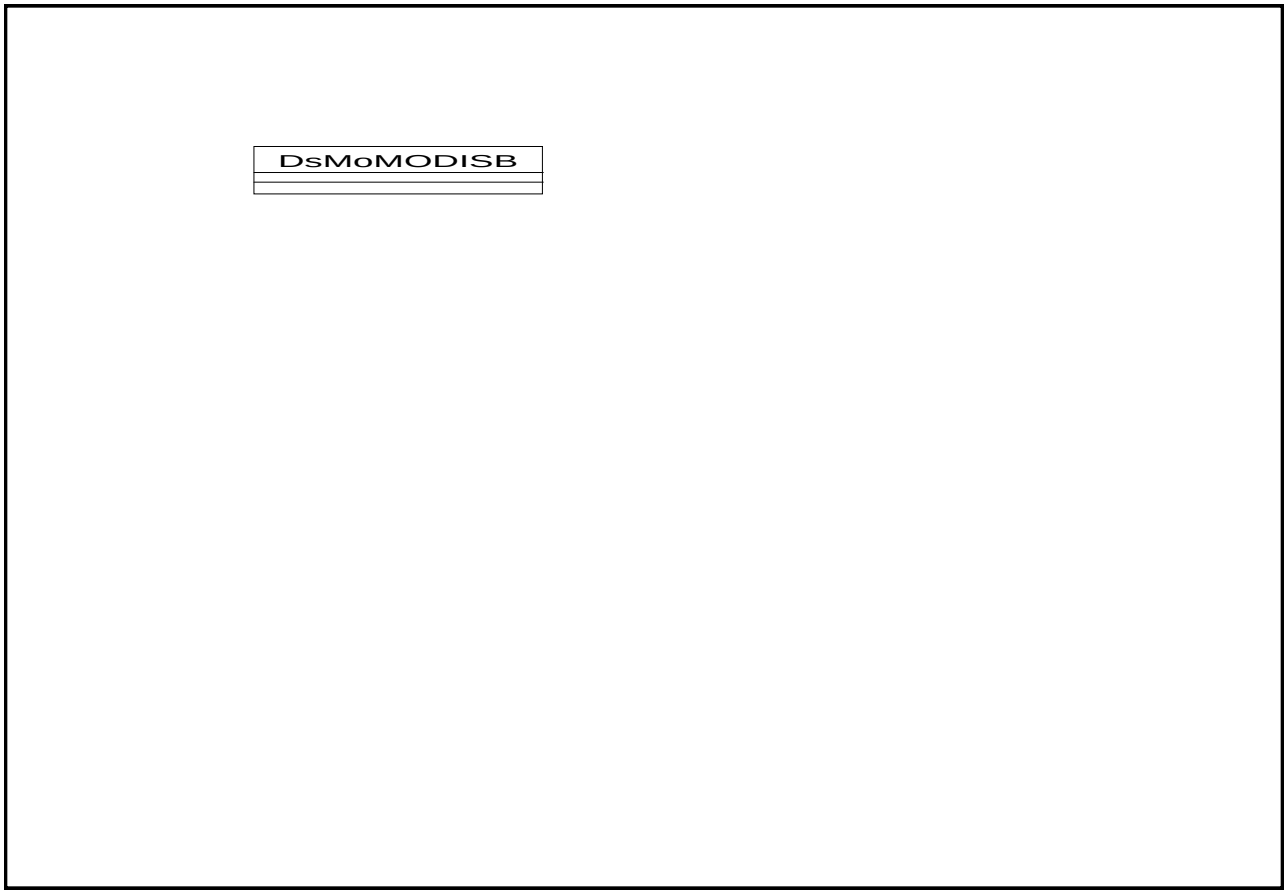


Figure 37 Main

4.3.23.2 mo Classes

4.3.23.2.1 DsMoMODISB Class

Overview:

DIstributed Object: No This class is an abstraction of the products derived from the MODIS Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsMoMODISB();

~DsMoMODISB();
```

Operations:

4.3.24 mp Class Category

4.3.24.1 Overview

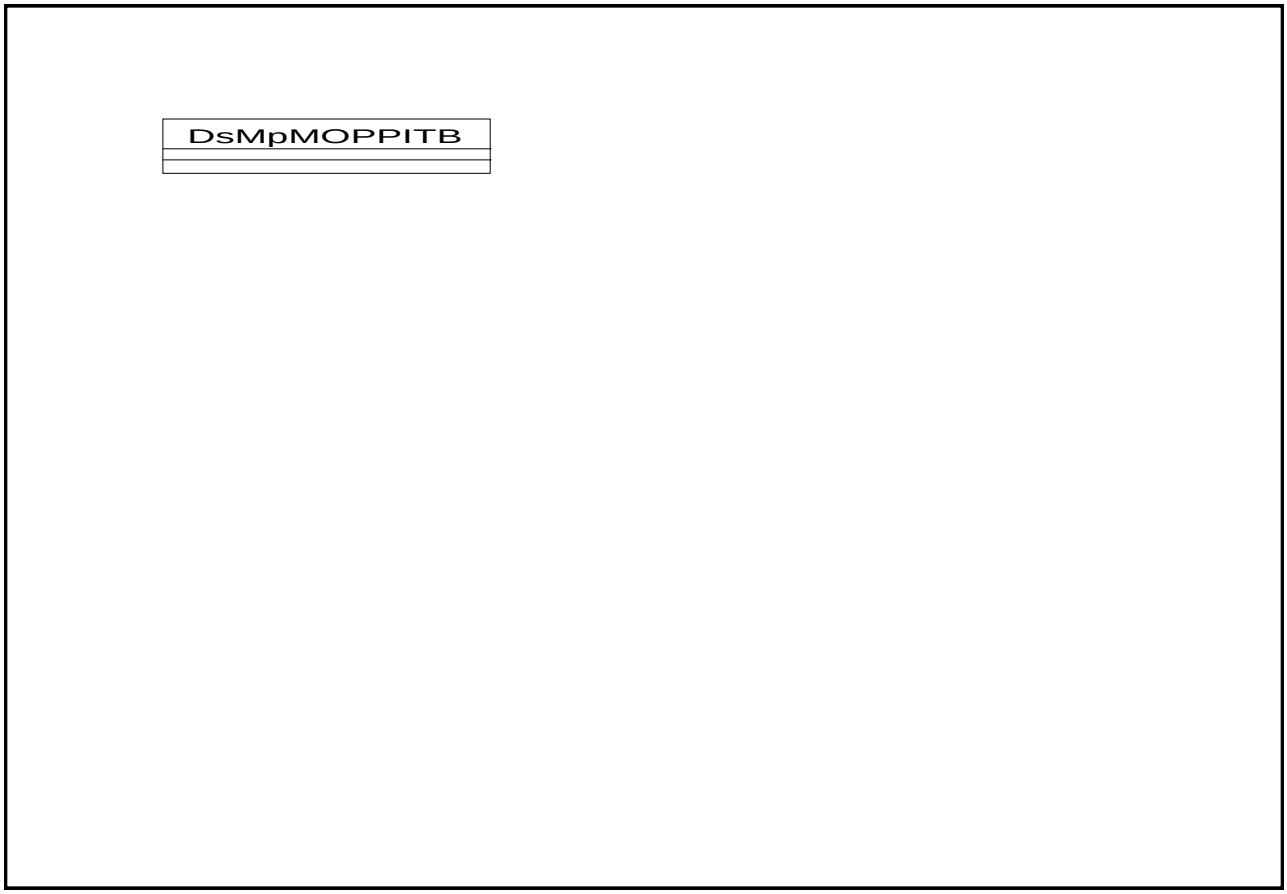


Figure 38 Main

4.3.24.2 mp Classes

4.3.24.2.1 DsMpMOPPITB Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the MOPPIT Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
~DsMpMOPPITB();
```

Operations:

- DsNoNOPPITB

```
DsNoNOPPITB();
```

4.3.25 ms Class Category

4.3.25.1 Overview

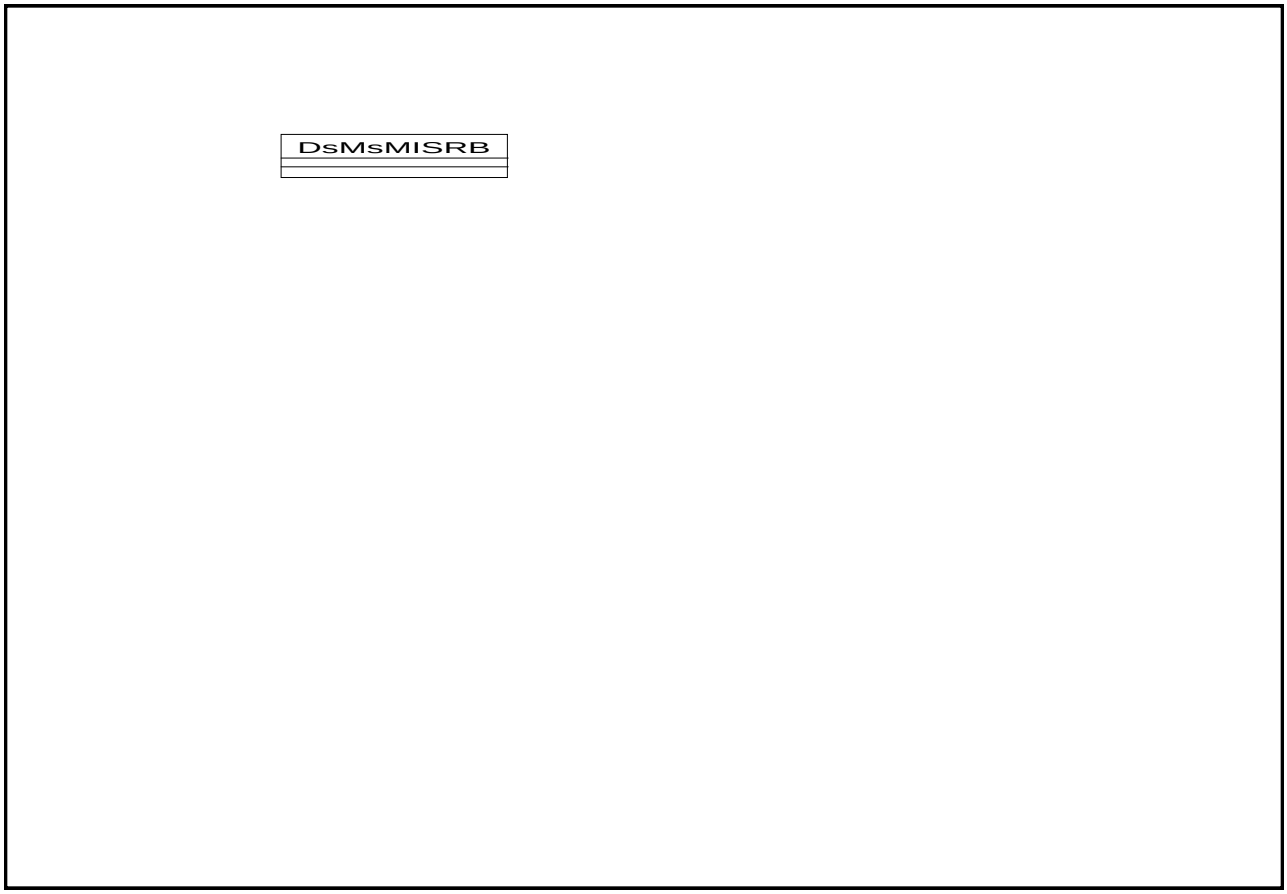


Figure 39 Main

4.3.25.2 ms Classes

4.3.25.2.1 DsMsMISRB Class

Overview:

DIstributed Object: No This class is an abstraction of the products derived from the TRMM CERES Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsMsMISRB();

~DsMsMISRB();
```

Operations:

4.3.26 nm Class Category

4.3.26.1 Overview

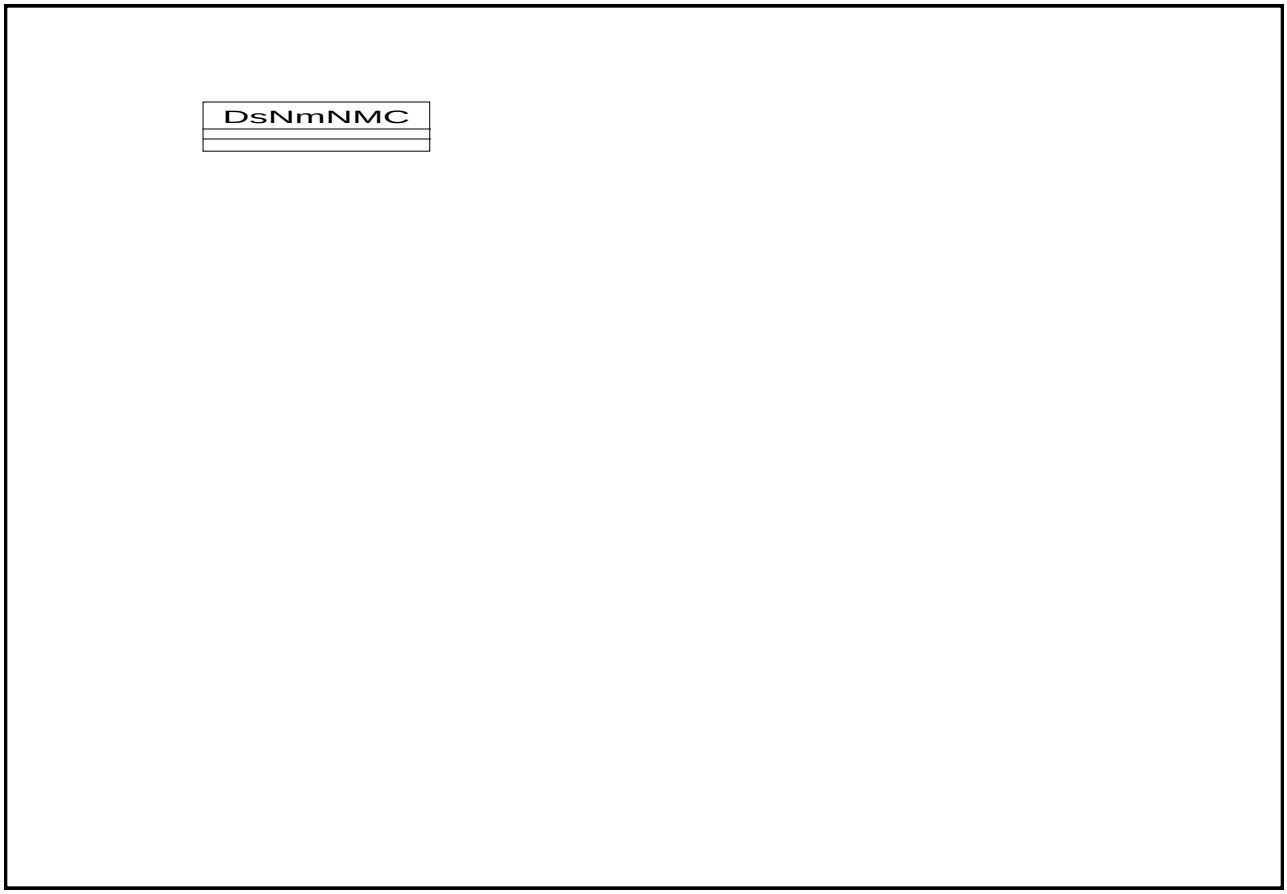


Figure 40 Main

4.3.26.2 nm Classes

4.3.26.2.1 DsNmNMC Class

Overview:

Distributed Object: No This class is an abstraction of the products from the National Meteorological Center.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

```
DsNmNMC();

~DsNmNMC();
```

Operations:

4.3.27 np Class Category

4.3.27.1 Overview

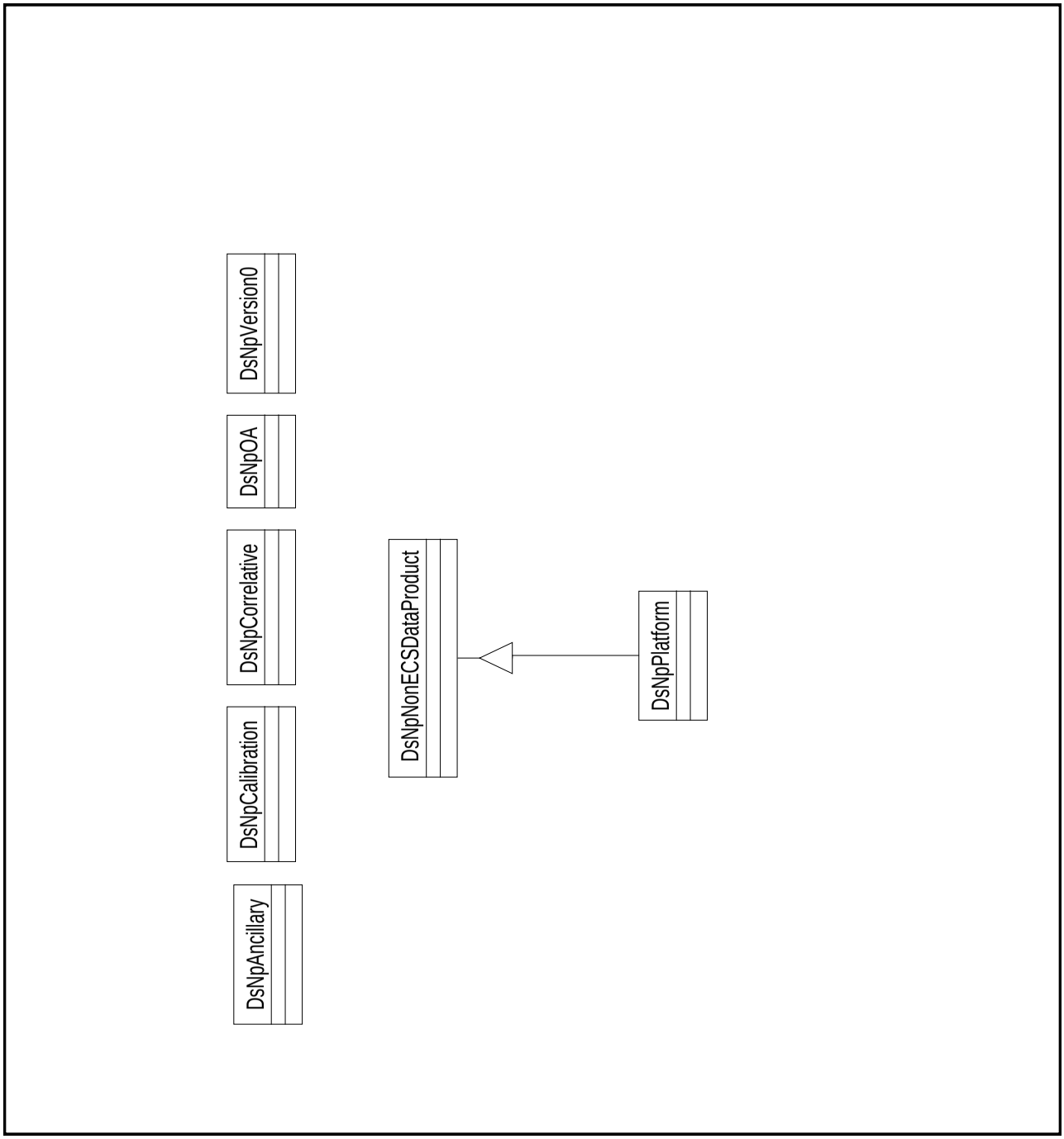


Figure 41 Main

4.3.27.2 np Classes

4.3.27.2.1 DsNpAncillary Class

Overview:

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.

Export Control: Public

Inheritance Relationships:

Inherits from DsNsNonECSDataProduct

Attributes:

Constructors and Destructor:

Operations:

4.3.27.2.2 DsNpCalibration Class

Overview:

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.

Export Control: Public

Inheritance Relationships:

Inherits from DsNsNonECSDataProduct

Attributes:

Constructors and Destructor:

Operations:

4.3.27.2.3 DsNpCorrelative Class

Overview:

This class contains data products that are used as correlative data to evaluate and validate EOS data products.

Export Control: Public

Inheritance Relationships:

Inherits from DsNsNonECSDataProduct

Attributes:

Constructors and Destructor:

Operations:

4.3.27.2.4 DsNpNonECSDataProduct Class

Overview:

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.

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.27.2.5 DsNpOA Class

Overview:

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.

Export Control: Public

Inheritance Relationships:

Inherits from DsNsNonECSDataProduct

Attributes:

Constructors and Destructor:

Operations:

4.3.27.2.6 DsNpPlatform Class

Overview:

This class represents Satellite Housekeeping Data. The format need not be know to the Data Server.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

Operations:

4.3.27.2.7 DsNpVersion0 Class

Overview:

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.

Export Control: Public

Inheritance Relationships:

Inherits from DsNsNonECSDataProduct

Attributes:

Constructors and Destructor:

Operations:

4.3.28 ns Class Category

4.3.28.1 Overview

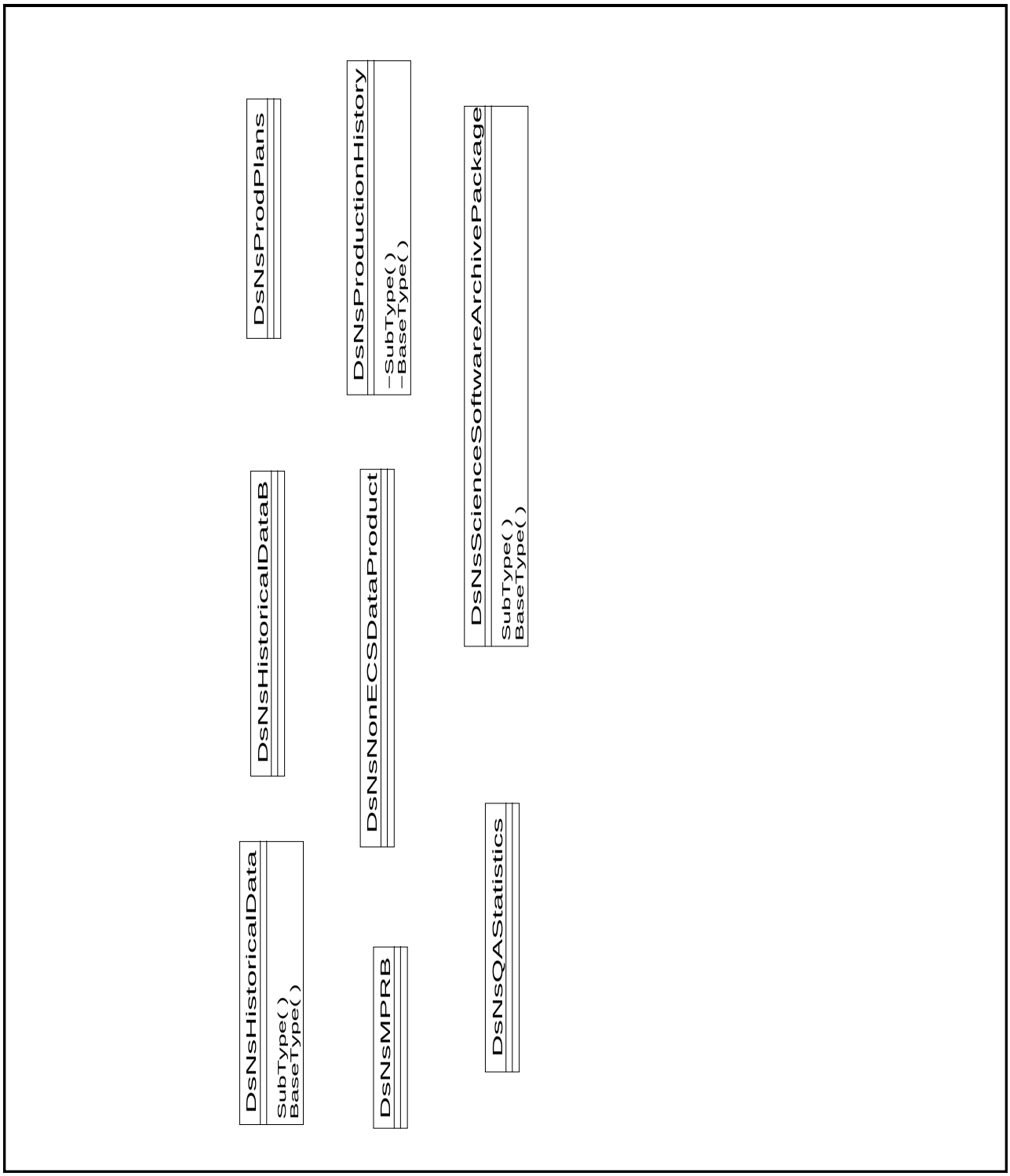


Figure 42 Main

4.3.28.2 ns Classes

4.3.28.2.1 DsNsHistoricalDataB Class

Overview:

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 this data type are n ot interpreted but merely archived and made available for distribution.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeESDT

Inherits from DsGeESDT

Attributes:

Constructors and Destructor:

Operations:

4.3.28.2.2 DsNsHistoricalData Class

Overview:

This file becomes a dynamic library that implements the Historical Data temporary data type. The "standard" entry point DsGeNewESDT is provided by all libraries.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeScienceData

Attributes:

Constructors and Destructor:

```
DsNsHistoricalData();
```

```
~DsNsHistoricalData();
```

default constructor BEGIN_PROLOG

NAME ~DsNsHistoricalData

DESCRIPTION Destructor for this class

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- BaseType

```
enum DsGeOID::_basetypetag BaseType() const;
```

Returns type of this instance BEGIN_PROLOG

NAME BaseType

DESCRIPTION This function returns the basetype name of this ESDT instance.

OUTPUTS None

RETURNS DsEGeBaseType - enum defined by GeOID class.

GLOBALS None

FILES None

END_PROLOG

- SubType

```
RWCString SubType() const;
```

BEGIN_PROLOG

NAME SubType

DESCRIPTION This function returns the datatype name of this ESDT instance.

OUTPUTS None

RETURNS An RWCString with the name of this ESDT instance.

GLOBALS None

FILES None

END_PROLOG

4.3.28.2.3 DsNsMPRB Class

Overview:

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeESDT

Attributes:

Constructors and Destructor:

Operations:

4.3.28.2.4 DsNsNonECSDataProduct Class

Overview:

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeScienceData

Attributes:

Constructors and Destructor:

Operations:

4.3.28.2.5 DsNsProdPlans Class

Overview:

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeESDT

Attributes:

Constructors and Destructor:

Operations:

4.3.28.2.6 DsNsProductionHistory Class

Overview:

This file becomes a dynamic library that implements the ProductionHistory temporary data type. The "standard" entry point DsGeNewESDT is provided by all libraries.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeScienceData

Attributes:

Constructors and Destructor:

```
DsNsProductionHistory();
```

```
~DsNsProductionHistory();
```

default constructor BEGIN_PROLOG

NAME ~DsNsProductionHistory

DESCRIPTION Destructor for this class

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- BaseType

```
enum DsGeOID::_basetypetag BaseType() const;
```

Returns type of this instance BEGIN_PROLOG

NAME BaseType

DESCRIPTION This function returns the basetype name of this ESDT instance.

OUTPUTS None

RETURNS DsEGeBaseType - enum defined by GeOID class.

GLOBALS None

FILES None

END_PROLOG

- SubType

```
RWCString SubType() const;
```

BEGIN_PROLOG

NAME SubType

DESCRIPTION This function returns the datatype name of this ESDT instance.

OUTPUTS None

RETURNS An RWCString with the name of this ESDT instance.

GLOBALS None

FILES None

END_PROLOG

4.3.28.2.7 DsNsQAStatistics Class

Overview:

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeESDT

Attributes:

Constructors and Destructor:

Operations:

4.3.28.2.8 DsNsScienceSoftwareArchivePackage Class

Overview:

This file becomes a dynamic library that implements the ScienceSoftware ArchivePackage temporary data type. The "standard" entry point DsGeNewESDT is provided by all libraries.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsGeScienceData

Attributes:

Constructors and Destructor:

```
DsNsScienceSoftwareArchivePackage();
```

~DsNsScienceSoftwareArchivePackage();

default constructor BEGIN_PROLOG

NAME ~DsNsScienceSoftwareArchivePackage

DESCRIPTION Destructor for this class

OUTPUTS None

RETURNS N/A

GLOBALS None

FILES None

END_PROLOG

Operations:

- BaseType

  enum DsGeOID::_basetypetag BaseType() const;

  Returns type of this instance BEGIN_PROLOG

  NAME BaseType

  DESCRIPTION This function returns the basetype name of this ESDT instance.

  OUTPUTS None

  RETURNS DsEGeBaseType - enum defined by GeOID class.

  GLOBALS None

  FILES None

  END_PROLOG

- SubType

  RWCString SubType() const;

  BEGIN_PROLOG

  NAME SubType

  DESCRIPTION This function returns the datatype name of this ESDT instance.

  OUTPUTS None

  RETURNS An RWCString with the name of this ESDT instance.

  GLOBALS None

  FILES None

  END_PROLOG

4.3.29 pr Class Category

4.3.29.1 Overview

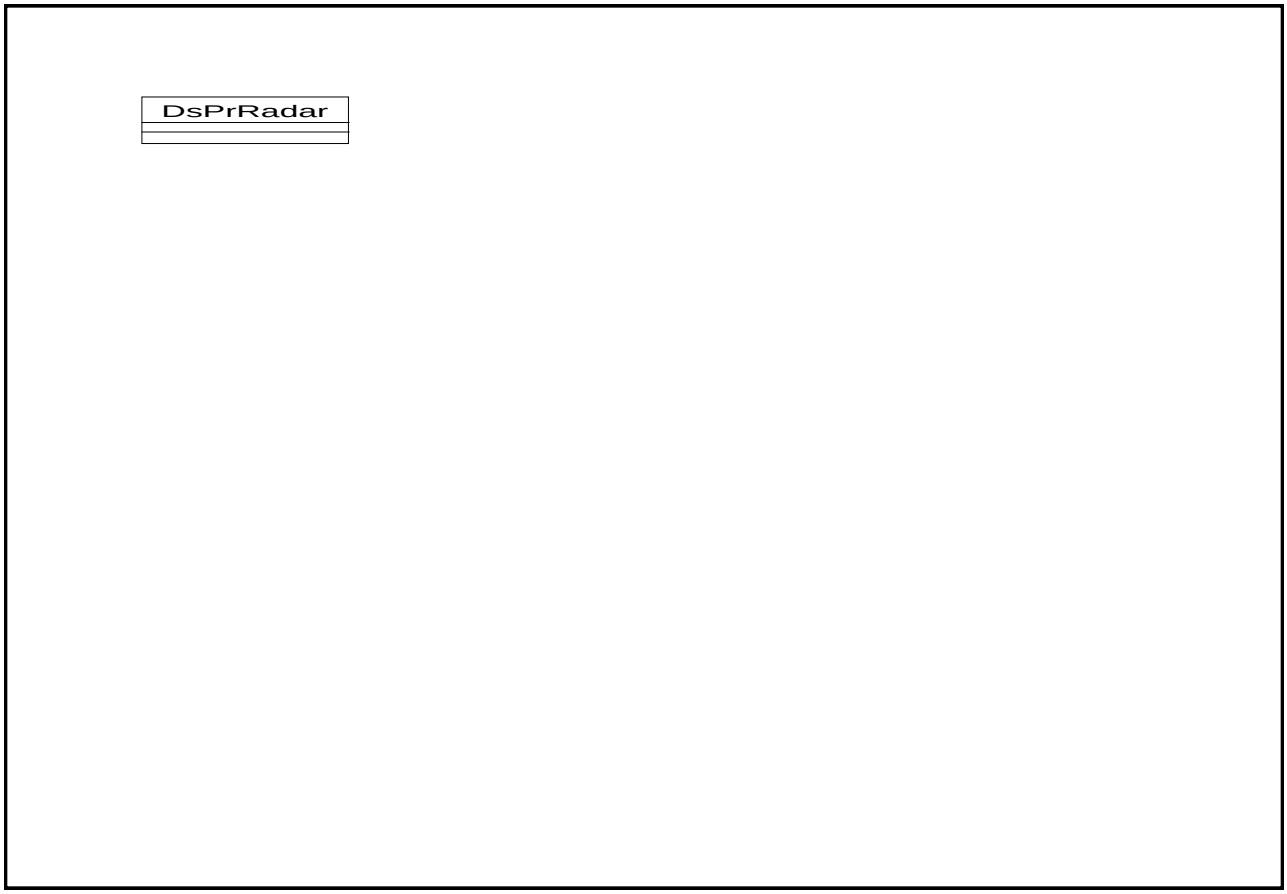


Figure 43 Main

4.3.29.2 pr Classes

4.3.29.2.1 DsPrRadar Class

Overview:

Distributed Object: No The class is an anstraction of the products from the TRMM platform Precipitation Radar (PR) instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

```
DsPrRadar();

~DsPrRadar();
```

Operations:

4.3.30 ra Class Category

4.3.30.1 Overview

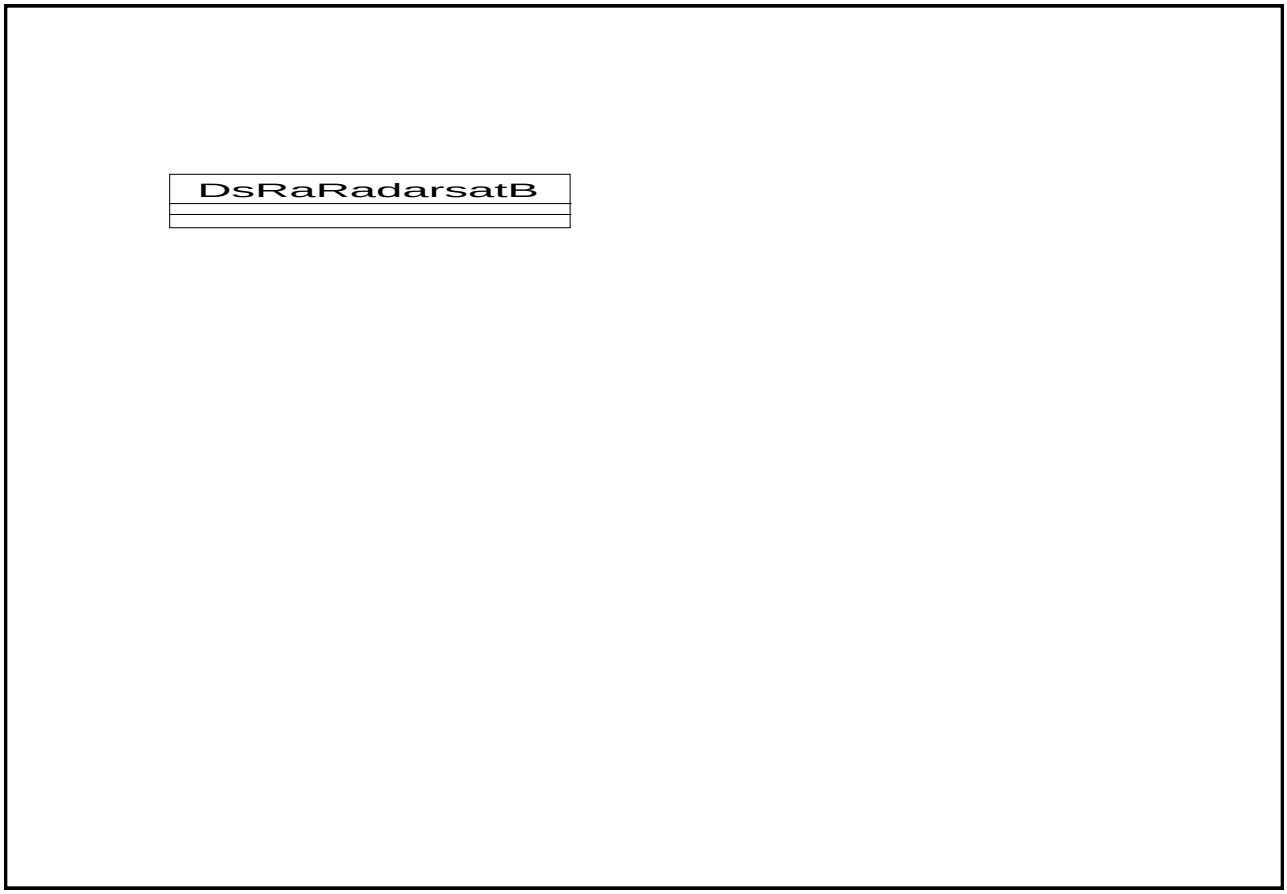


Figure 44 Main

4.3.30.2 ra Classes

4.3.30.2.1 DsRaRadarsatB Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the instruments on the Radarsat satellite.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsRaRadarsatB();

~DsRaRadarsatB();
```

Operations:

4.3.31 sa Class Category

4.3.31.1 Overview

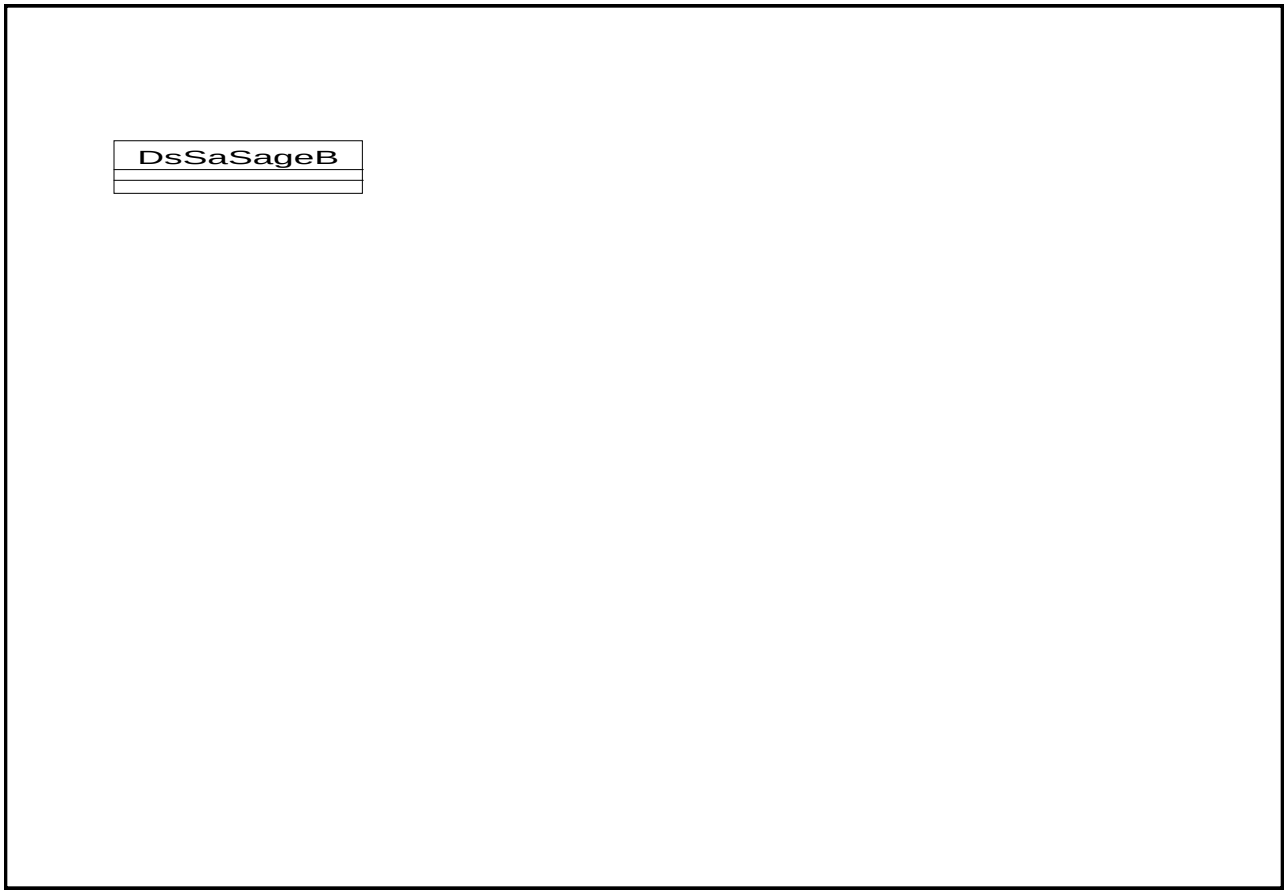


Figure 45 Main

4.3.31.2 sa Classes

4.3.31.2.1 DsSaSageB Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the Sage Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsSaSageB();

~DsSaSageB();
```

Operations:

4.3.32 sh Class Category

4.3.32.1 Overview

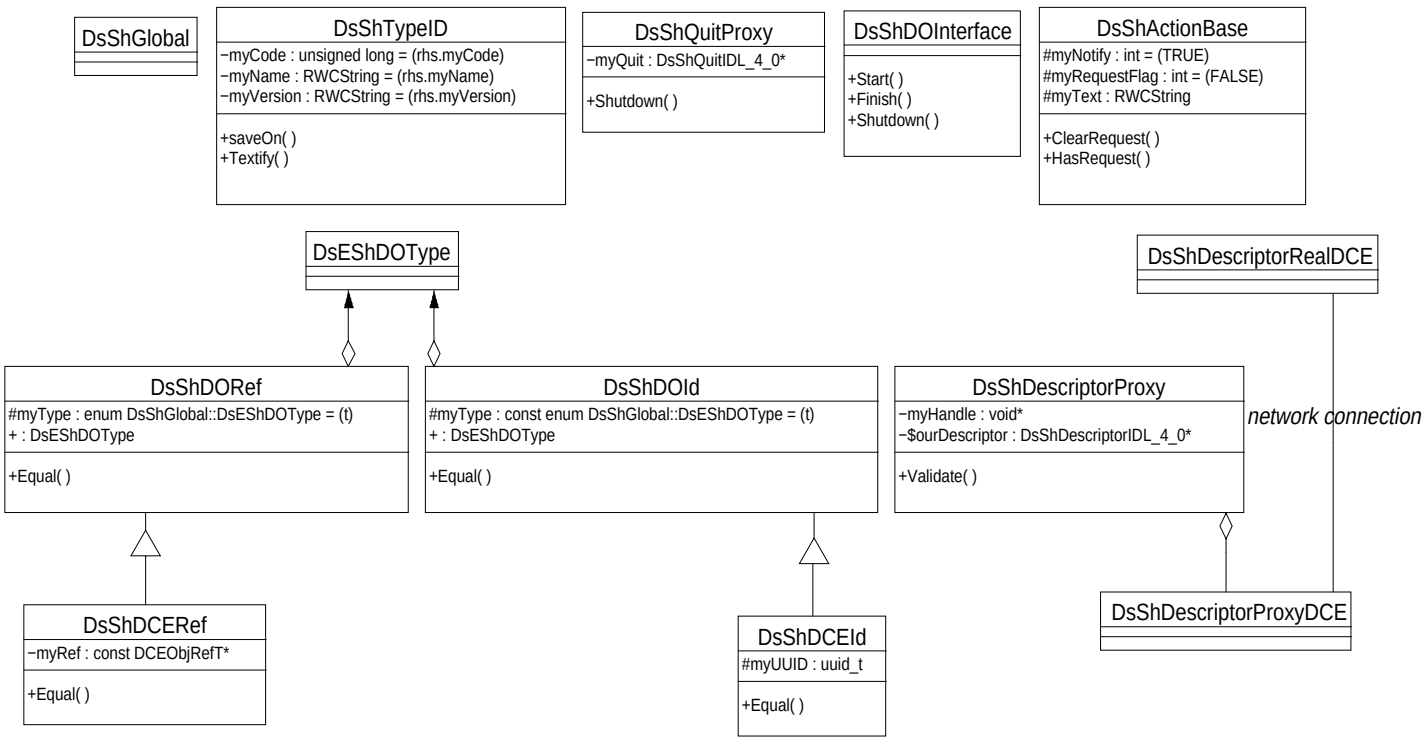


Figure 40 DsSh1

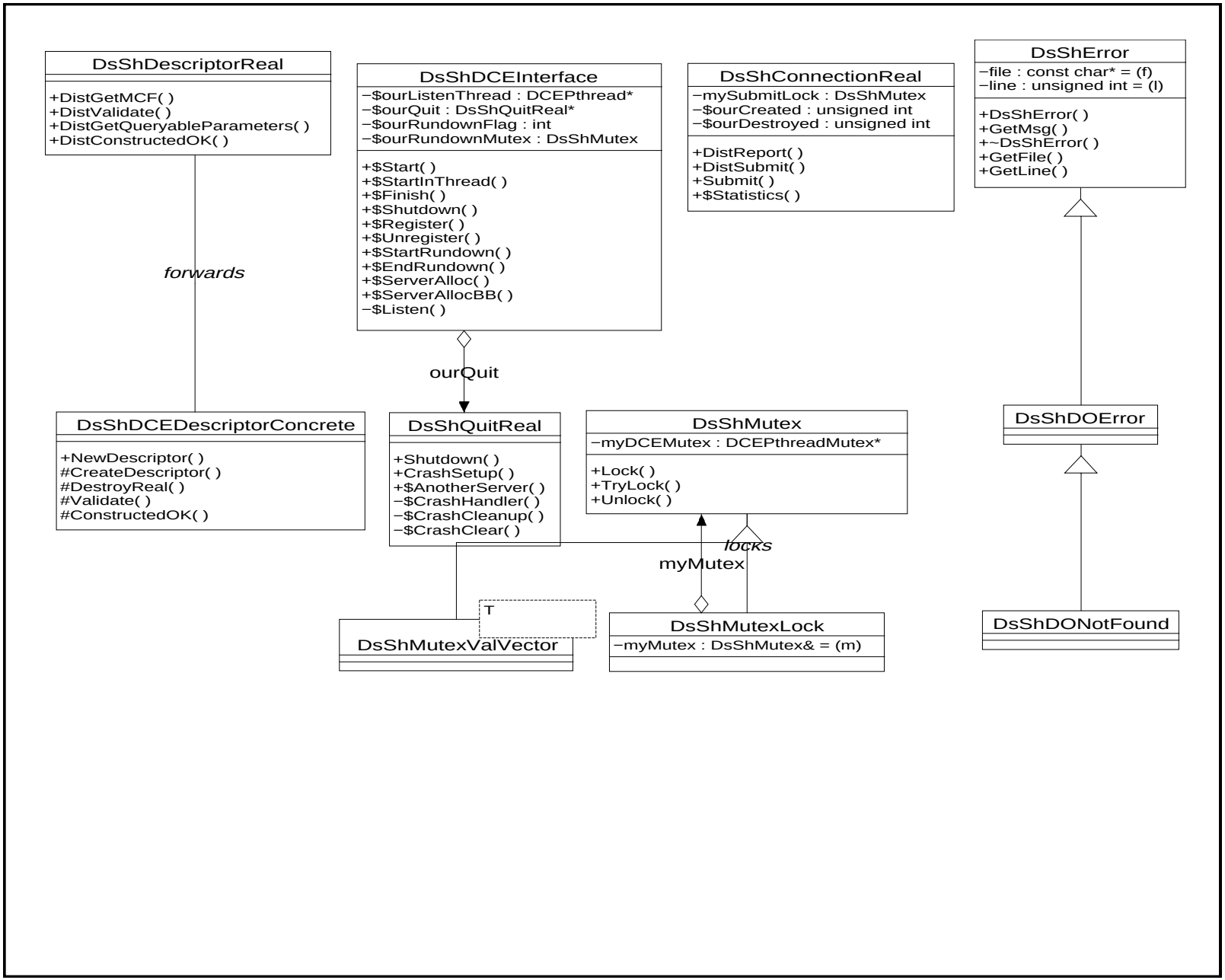


Figure 4: DsSh2

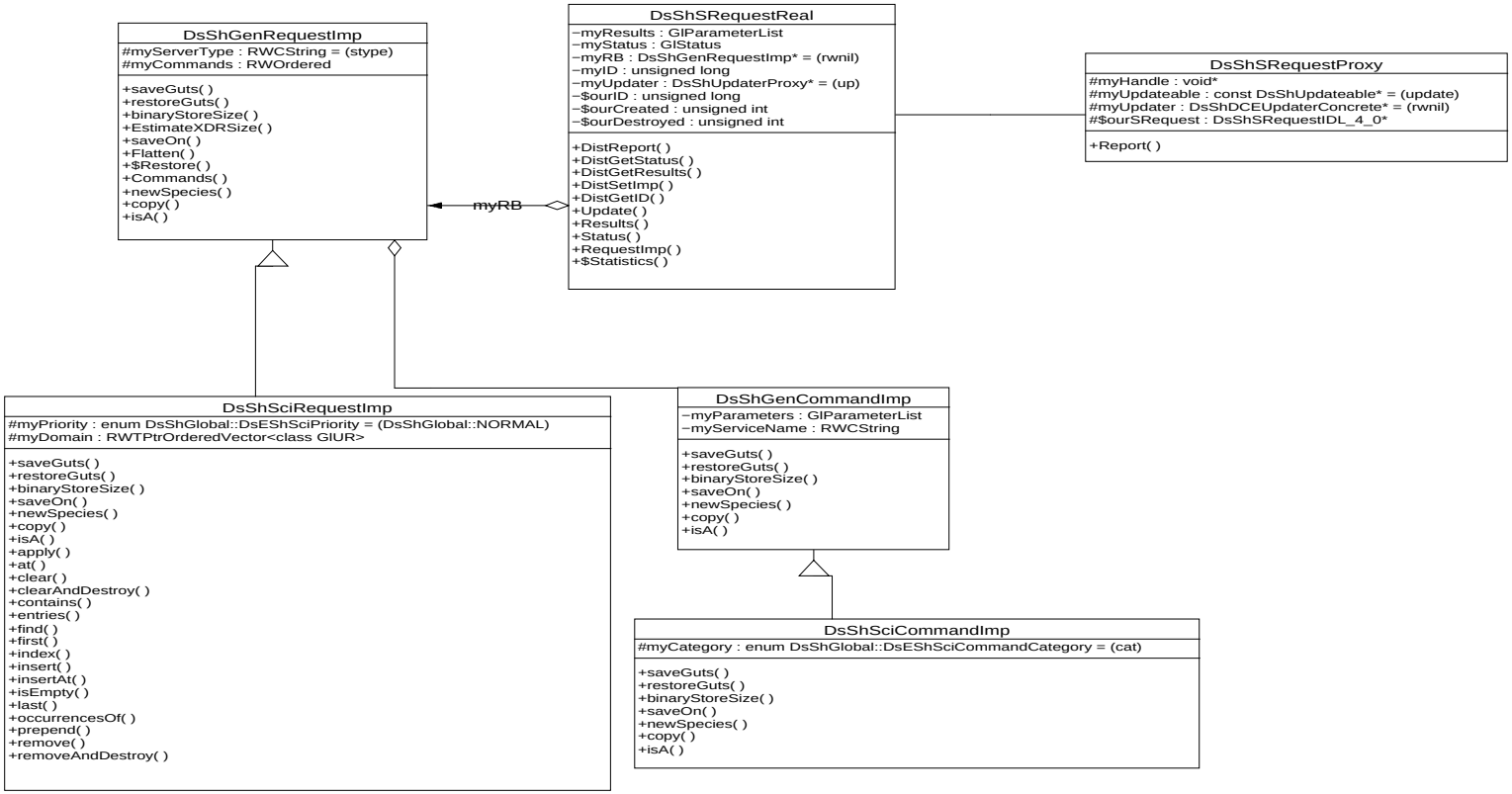
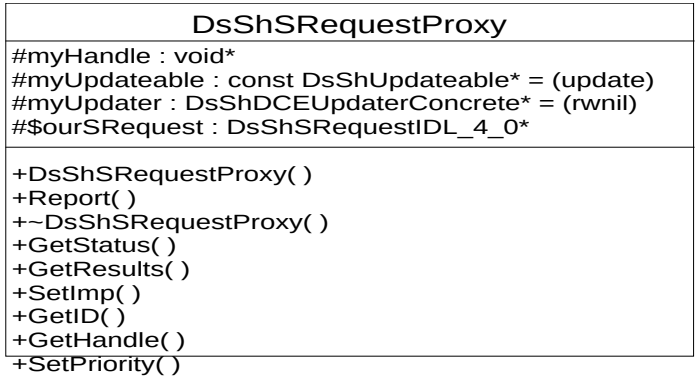


Figure 48 DsSh3



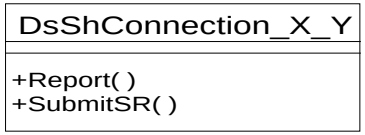


Figure 49 DsSh4

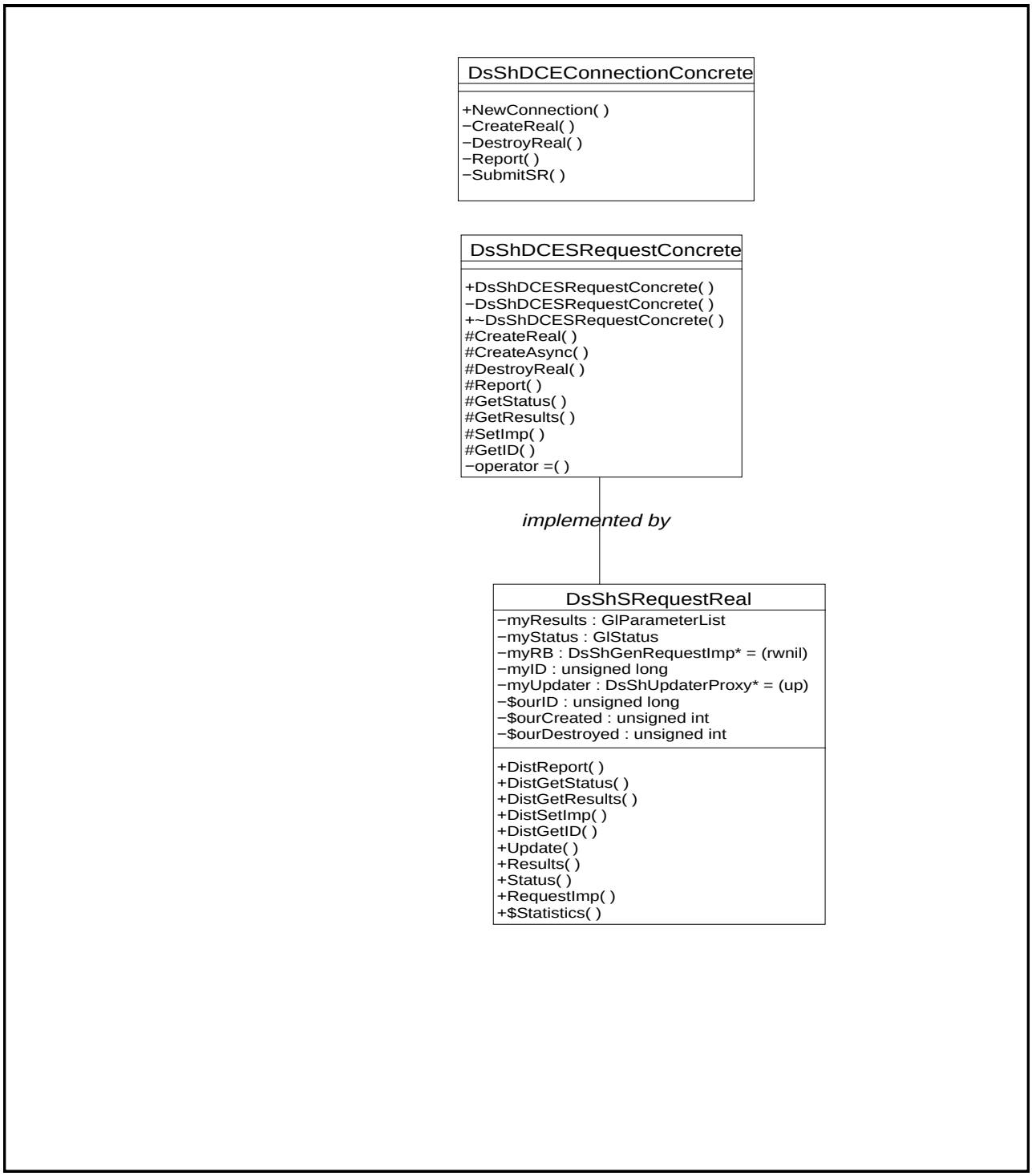


Figure 50 DsSh5

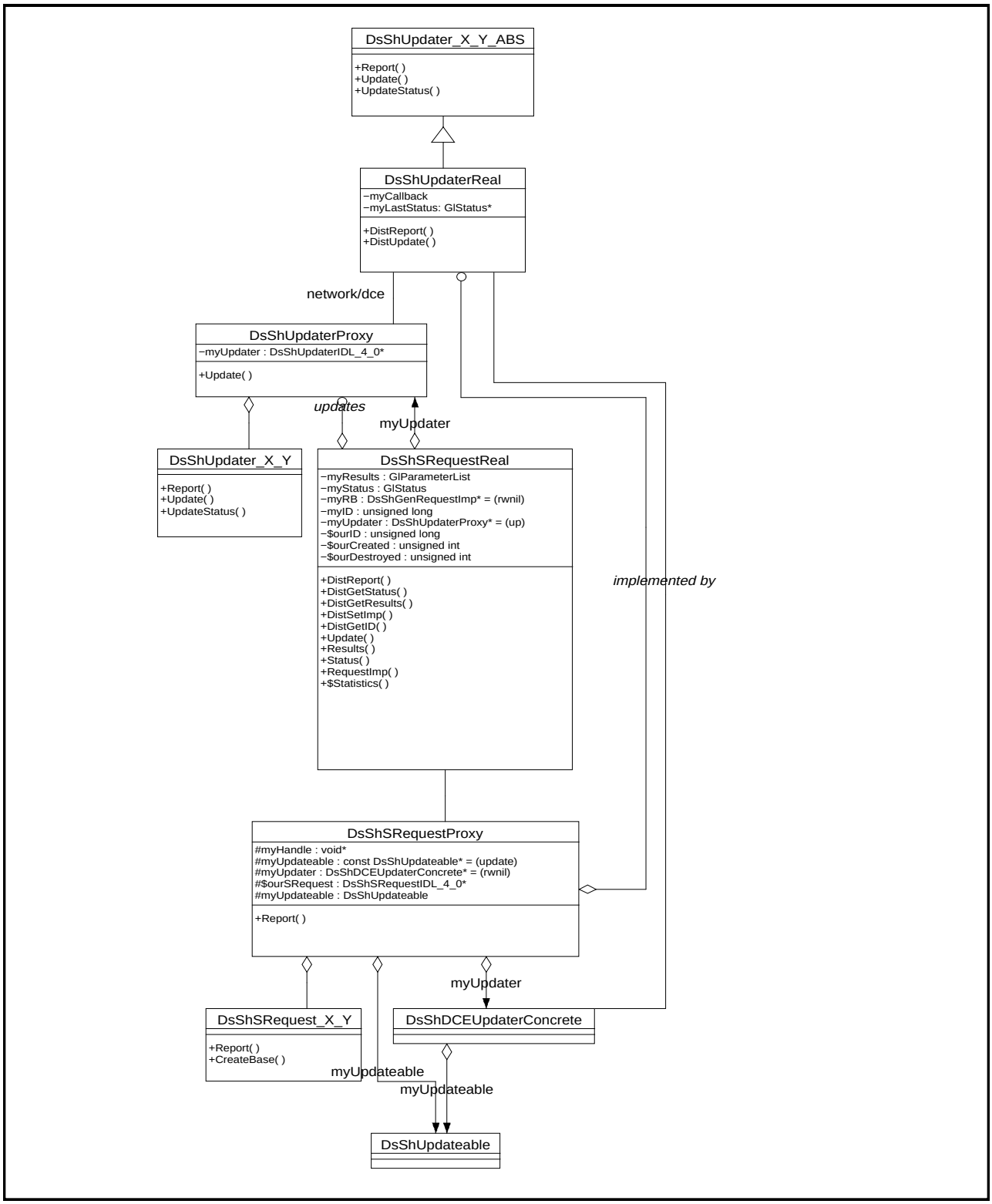


Figure 51 DsSh6

4.3.32.2 sh Classes

4.3.32.2.1 DsEShDOType Class

Overview:

these typedefs allow the type names (which have the ECS prefixes) to be used outside the DsShGlobal scope, but doesn't expose the enum values to the global scope Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.32.2.2 DsShActionBase Class

Overview:

Declaration of the base class for actions to be performed when subscriptions fire. Implementation of subscriptions is phase two.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myNotify: int

myRequestFlag: int
```

whether to notify on firing

`myText: RWCString`

whether request has been set

Constructors and Destructor:

`DsShActionBase(int notify, const RWCString* text);`

normal constructor

`DsShActionBase();`

hidden default constructor

`virtual  ~DsShActionBase();`

notification text destructor BEGIN_PROLOG

NAME ~DsShActionBase

DESCRIPTION destructor

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- ClearRequest

  `virtual void ClearRequest() = 0;`

  clear any request that has been set

- GetNotify

  `int GetNotify() const;`

  new value for the notify flag obtain current value of notify flag BEGIN_PROLOG

  NAME GetNotify

  DESCRIPTION Obtain the value of the notify flag

  RETURNS the current value of the notify flag

  GLOBALS none

  FILES none

  END_PROLOG

- GetText

  `const RWCString& GetText() const;`

obtain notification text

- HasRequest

  ```
  int HasRequest() const;
  ```

  find out if any request has been set

- SetNotify

  ```
  void SetNotify(int notify);
  ```

  set the notify flag BEGIN_PROLOG

  NAME SetNotify

  DESCRIPTION Set a new value for the notify flag.

  RETURNS nothing

  GLOBALS none

  FILES none

  END_PROLOG

- SetText

  ```
  void SetText(RWCString& text);
  ```

  set notification text

4.3.32.2.3 DsShConnectionReal Class

Overview:

Declarations for the class DsShConnectionReal. This class provides a server interface to the server's connection distributed object. It inherits from the DCE-generated server connection class, and adds functions to provide stronger type-checking.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
mySubmitLock: DsShMutex
```

```
ourCreated: unsigned int
```

the one-submit-at-a-time per server lock

`ourDestroyed: unsigned int`

number of requests created

`ourSubmitLock: DsShMutex`

controls DistSubmit threading

Constructors and Destructor:

`DsShConnectionReal();`

default constructor number of connections destroyed BEGIN_PROLOG

NAME DsShConnectionReal

DESCRIPTION default constructor. Initializes the base classes, and logs the connection creation (and counts it).

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`virtual  ~DsShConnectionReal();`

destructor BEGIN_PROLOG

NAME ~DsShConnectionReal

DESCRIPTION destructor

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- DistReport

`void DistReport() const;`

DISTRIBUTED FUNCTION IMPLEMENTATIONS log a message identifying this request BEGIN_PROLOG

NAME Report

DESCRIPTION logs an identifying message on the server

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- DistSubmit

```
void DistSubmit(DsShSRequestReal* thesr);
```

submit a request BEGIN_PROLOG

NAME SubmitSR

DESCRIPTION The distributed request-submission function. Performs the request-level locking, which means that only one request (total across all connections and all clients) can be active within the server at a time. [Later phases should remove this restriction]. Also locks within this connection, which is redundant with the request-level locking, but will be used when the request-level locking is removed.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- Statistics

```
static void Statistics();
```

dump static request statistics to the log BEGIN_PROLOG

NAME Statistics

DESCRIPTION Dumps statistics about connections created and destroyed. Generally used when server is shutting down.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- Submit

```
virtual void Submit(DsShSRequestReal* );
```

LOCAL FUNCTIONS virtual submit BEGIN_PROLOG

NAME Submit

DESCRIPTION higher-level submit function that accepts a request directly. This virtual function is expected to be overridden by derived classes to handle each request.

RETURNS status

GLOBALS none

FILES none

END_PROLOG

4.3.32.2.4 DsShConnection_X_Y Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

- Report

```
Report();
```

- SubmitSR

```
SubmitSR();
```

4.3.32.2.5 DsShDCEConnectionConcrete Class

Overview:

Declartions for the DsShDCEConnectionConcrete class. This class encapsulates all DCE exposure for the connection objects on the server-side.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsShDCEConnectionConcrete();
```

default constructor

```
DsShDCEConnectionConcrete(const DsShDCEConnectionConcrete& );
```

prevent copying or assigning

```
virtual  ~DsShDCEConnectionConcrete();
```

destructor BEGIN_PROLOG
NAME ~DsShDCEConnectionConcrete
DESCRIPTION
RETURNS nothing
END_PROLOG

Operations:

- CreateReal

```
DsTShConnectionHandle CreateReal();
```

DISTRIBUTED FUNCTIONS create a new one BEGIN_PROLOG
NAME Create synchronous real
DESCRIPTION used to create a new real
RETURNS handle to new real
END_PROLOG

- DestroyReal

```
void DestroyReal(DsTShConnectionHandle* handle);
```

destroy one BEGIN_PROLOG
NAME DestroyReal
DESCRIPTION delete a real
RETURNS nothing
END_PROLOG

- GetReal

```
DsShConnectionReal* GetReal(DsTShConnectionHandle ) const;
```

obtain the associated DCE-independent connection

- NewConnection

  ```
  virtual DsShConnectionReal* NewConnection();
  ```

  subclasses can override this BEGIN_PROLOG

  NAME NewConnection

  DESCRIPTION create a new connection. Most useful servers will subclass DsShDCEConnectionConcrete and override this function.

  RETURNS a real connection

  END_PROLOG

- operator =

  ```
  DsShDCEConnectionConcrete& operator =(const
  DsShDCEConnectionConcrete& );
  ```

- Report

  ```
  void Report(DsTShConnectionHandle handle);
  ```

  log a message identifying this request BEGIN_PROLOG

  NAME Report

  DESCRIPTION Used to log the existence and identity of this connection.

  RETURNS nothing

  END_PROLOG

- SubmitSR

  ```
  void SubmitSR(DsTShConnectionHandle );
  ```

  submit a request to the server

4.3.32.2.6 DsShDCEDescriptorConcrete Class

Overview:

forward declarations Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsShDCEDescriptorConcrete();
```

default constructor

```
DsShDCEDescriptorConcrete(const DsShDCEDescriptorConcrete& );
```

copy ctor, NOT IMPLEMENTED

```
virtual  ~DsShDCEDescriptorConcrete();
```

destructor BEGIN_PROLOG

NAME ~DsShDCEDescriptorConcrete

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- ConstructedOK

```
DsTShByteBuffer* ConstructedOK(DsTShDescriptorHandle handle);
```

I/O. Queryable parameters BEGIN_PROLOG

NAME ConstructedOK

DESCRIPTION Determine whether the descriptor was created successfully.

RETURNS Buffer containing status of object

END_PROLOG

- CreateDescriptor

```
DsTShDescriptorHandle CreateDescriptor(DsTShStringIDL name,
DsTShStringIDL vers);
```

DISTRIBUTED FUNCTIONS make a new real BEGIN_PROLOG

NAME Create synchronous real

DESCRIPTION used to create a new real

RETURNS handle to new real

END_PROLOG

- DestroyReal

```
void DestroyReal(DsTShDescriptorHandle* handle);
```

destroy a real BEGIN_PROLOG

NAME DestroyReal

DESCRIPTION delete a real

RETURNS nothing

END_PROLOG

- GetMCF

```
DsTShByteBuffer* GetMCF(DsTShDescriptorHandle handle,
DsTShByteBuffer** buffpp);
```

get metadata configuration file BEGIN_PROLOG

NAME GetMCF

DESCRIPTION

RETURNS Buffer which has a flattened GlStatus object indicating success or failure

END_PROLOG

- GetQueryableParameters

```
DsTShByteBuffer* GetQueryableParameters(DsTShDescriptorHandle
handle, DsTShByteBuffer** buffpp);
```

Input. Metadata to validate obtain queryable parameters BEGIN_PROLOG

NAME GetQueryableParameters

DESCRIPTION

RETURNS Buffer which has a flattened GlStatus object indicating success or failure

END_PROLOG

- NewDescriptor

```
virtual DsShDescriptorReal* NewDescriptor(const RWCString& name,
const RWCString& vers) = 0;
```

create a new real

- operator =

```
DsShDCEDescriptorConcrete& operator =(const
DsShDCEDescriptorConcrete& );
```

assignment operator, NOT IMPLEMENTED

- Validate

```
DsTShByteBuffer* Validate(DsTShDescriptorHandle handle,
DsTShByteBuffer** buffpp);
```

output stream Validate metadata BEGIN_PROLOG

NAME Validate

DESCRIPTION

RETURNS Buffer which has a flattened GlStatus object indicating success or failure

END_PROLOG

4.3.32.2.7 DsShDCEId Class

Overview:

Declarations for the class DsShDCEId. This class encapsulates the DCE object identifier, uuid_t. It can be used in a DCE-independent manner by reference to its base class (DsShDOId)

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShDOId

Attributes:

```
myUUID: uuid_t
```

Constructors and Destructor:

```
DsShDCEId(const uuid_t& u);
```

normal constructor

```
DsShDCEId(const DsShDCEId& );
```

copy constructor NOT IMPLEMENTED

```
virtual  ~DsShDCEId();
```

destructor

Operations:

- Equal

```
virtual int Equal(const DsShDOId& rhs) const;
```

inherited comparison function

- GetId

```
const uuid_t& GetId() const;
```

obtain the underlying DCE id

- operator =

  `DsShDCEId& operator =(const DsShDCEId& );`

  assignment NOT IMPLEMENTED

4.3.32.2.8 DsShDCEInterface Class

Overview:

Declarations for the DsShDCEInterface class. This class encapsultes DCE implementations of DO interface functions, and adds other functions needed by other DCE implementation objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`ourListenThread: DCEPthread*`

`ourQuit: DsShQuitReal*`

listen thread

`ourRundownFlag: int`

quit object

`ourRundownMutex: DsShMutex`

flag that protects theServer during rundowns

Constructors and Destructor:

`DsShDCEInterface();`

prevent construction

Operations:

- EndRundown

```
static void EndRundown();
```

required just before rundown returns BEGIN_PROLOG

NAME EndRundown

DESCRIPTION A DCE-specific function that is called to signal the end of the rundown procedure.

RETURNS nothing

END_PROLOG

- Finish

```
static void Finish();
```

call to finish DCE cleanup BEGIN_PROLOG

NAME Stop

DESCRIPTION Called by the main program when the Start call returns. Performs DCE cleanup.

RETURNS nothing

END_PROLOG

- Listen

```
static DCEPthreadResult Listen();
```

used by StartInThread, never called by users

- Register

```
static void Register(const DCEObj* obj);
```

global DCE object registration BEGIN_PROLOG

NAME Register

DESCRIPTION A DCE-specific function that registers DCE objects for proper connection & cleanup.

RETURNS Only if there's a startup problem, or the server is shut down.

END_PROLOG

- ServerAllocBB

```
static DsTShByteBuffer* ServerAllocBB(unsigned int bytecount);
```

Use this one to allocate a bytebuffer that'll hold the given number of bytes BEGIN_PROLOG

NAME ServerAllocBB

DESCRIPTION A DCE-specific function that allocates memory for a server to use that can be returned to a client. DCE is responsible for freeing this memory when the client return successfully finishes.

RETURNS a newly allocated (via DCE) bytebuffer

END_PROLOG

- ServerAlloc

  `static void* ServerAlloc(unsigned int bytecount);`

  Use these functions on a server to allocate memory that is returned to a client (DCE will free the memory when the return to the client finishes). BEGIN_PROLOG

  NAME ServerAlloc

  DESCRIPTION A DCE-specific function that allocates memory for a server to use that can be returned to a client. DCE is responsible for freeing this memory when the client return successfully finishes.

  RETURNS pointer to the special memory

  END_PROLOG

- Shutdown

  `static void Shutdown();`

  called on server shutdown BEGIN_PROLOG

  NAME Shutdown

  DESCRIPTION Called on server shutdown. Performs necessary DCE shutdown functions.

  RETURNS nothing

  END_PROLOG

- StartInThread

  `static int StartInThread();`

  start a listen loop in a new thread BEGIN_PROLOG

  NAME StartInThread

  DESCRIPTION starts a listen loop in a new thread

  RETURNS TRUE iff listen loop successfully started

  END_PROLOG

- StartRundown

  `static void StartRundown();`

  required before rundown does anything BEGIN_PROLOG

  NAME StartRundown

  DESCRIPTION A DCE-specific function that protects the other static functions from using theServer while a rundown procedure is active (which locks up the server)

  RETURNS nothing

  END_PROLOG

- Start

```
static void Start();
```

call to start listen loop BEGIN_PROLOG

NAME Start

DESCRIPTION Called on server startup. Performs all DCE functions necessary for startup, and enters the listen loop.

RETURNS Only if there's a startup problem, or the server is shut down.

END_PROLOG

- Unregister

```
static void Unregister(const DCEObj* obj);
```

global DCE object un-registration BEGIN_PROLOG

NAME Unregister

DESCRIPTION A DCE-specific function that un-registers DCE objects (presumeably because they are no longer needed by the client)

RETURNS nothing

END_PROLOG

4.3.32.2.9 DsShDCERef Class

Overview:

Declarations for the DsShDCERef class. This class encapsulates the DCE object reference type DCEObjRefT.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShDORef

Attributes:

```
myRef: const DCEObjRefT*
```

Constructors and Destructor:

```
DsShDCERef(const DCEObjRefT* r);
```

normal constructor

```
DsShDCERef(const DsShDCERef& );
```

copy constructor NOT IMPLEMENTED

```
virtual  ~DsShDCERef();
```

destructor

Operations:

- Equal

```
virtual int Equal(const DsShDORef& rhs) const;
```

comparison

- GetRef

```
const DCEObjRefT* GetRef() const;
```

obtain the underlying reference

- operator =

```
DsShDCERef& operator =(const DsShDCERef& );
```

assignment NOT IMPLEMENTED

- SetRef

```
void SetRef(DCEObjRefT* r);
```

set a value for the underlying reference

4.3.32.2.10 DsShDCESRequestConcrete Class

Overview:

Declarations for the class DsShDCESRequestConcrete. This server-side class encapsulates all of the DCE behavior and signatures for srequests, and communicates with an associated DsShSRequestReal in a "DCE-clean" manner.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsShDCESRequestConcrete();
```

default constructor

```
DsShDCESRequestConcrete(const DsShDCESRequestConcrete& );
```

prevent copying

```
~DsShDCESRequestConcrete();
```

destructor BEGIN_PROLOG
NAME ~DsShDCESRequestConcrete
RETURNS nothing
END_PROLOG

Operations:

- CreateAsync

```
DsTShSRequestHandle CreateAsync(DCEObjRefT* obj);
```

create a new one BEGIN_PROLOG
NAME Create asynchronous real
DESCRIPTION used to create a new real
RETURNS handle to new real
END_PROLOG

- CreateReal

```
DsTShSRequestHandle CreateReal();
```

DISTRIBUTED FUNCTIONS create a new one BEGIN_PROLOG
NAME Create synchronous real
DESCRIPTION used to create a new real
RETURNS handle to new real
END_PROLOG

- DestroyReal

```
void DestroyReal(DsTShSRequestHandle* handle);
```

destroy one BEGIN_PROLOG
NAME DestroyReal
DESCRIPTION delete a real
RETURNS nothing
END_PROLOG

- GetID

  ```
  unsigned long GetID(DsTShSRequestHandle handle);
  ```

  length of byte array return the request's ID BEGIN_PROLOG

  NAME GetID

  DESCRIPTION returns the dataserver (not DCE) id number for this request

  RETURNS the id

  END_PROLOG

- GetResults

  ```
  void GetResults(DsTShSRequestHandle handle, DsTShByteBuffer**
  buffpp);
  ```

  return the current value of the results BEGIN_PROLOG

  NAME GetResults

  DESCRIPTION Flattens the current value of the results list and returns it to the client. At this level, obtains the GlParameterList from the real request, and performs the flattening/RPC stuff.

  RETURNS nothing

  END_PROLOG

- GetStatus

  ```
  void GetStatus(DsTShSRequestHandle handle, DsTShByteBuffer**
  buffpp);
  ```

  return the current status BEGIN_PROLOG

  NAME GetStatus

  DESCRIPTION Flattens the current value of the status and returns it to the client.

  RETURNS nothing

  END_PROLOG

- operator =

  ```
  DsShDCESRequestConcrete& operator =(const
  DsShDCESRequestConcrete& );
  ```

- Report

  ```
  void Report(DsTShSRequestHandle handle);
  ```

  log a message on the server identifying this request BEGIN_PROLOG

  NAME Report

  DESCRIPTION logs an identifying message on the server

RETURNS nothing

END_PROLOG

- SetImp

```
void SetImp(DsTShSRequestHandle handle, idl_byte* thereq,
idl_long_int len);
```

passes the core request information from the client to me BEGIN_PROLOG

NAME SetImp

DESCRIPTION Takes a flattened DsShGenRequestImp object, and reconstitutes it. Used by clients to transfer the "core" information of a request to the server before submitting the request for execution

RETURNS nothing

END_PROLOG

4.3.32.2.11 DsShDCEUpdaterConcrete Class

Overview:

Declartions for the DsShDCEUpdaterConcrete class. This class encapsulates all DCE exposure for the Updater objects on the server-side.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myUpdateable: DsShUpdateable&
```

Constructors and Destructor:

```
DsShDCEUpdaterConcrete(DsShUpdateable& up);
```

default constructor

```
DsShDCEUpdaterConcrete(const DsShDCEUpdaterConcrete& );
```

prevent copying or assigning

```
virtual  ~DsShDCEUpdaterConcrete();
```

destructor BEGIN_PROLOG

NAME ~DsShDCEUpdaterConcrete
DESCRIPTION
RETURNS nothing
END_PROLOG

Operations:

- operator =

```
DsShDCEUpdaterConcrete& operator =(const DsShDCEUpdaterConcrete&
);
```

- Update

```
void Update();
```

DISTRIBUTED FUNCTIONS perform an update BEGIN_PROLOG
NAME Update
DESCRIPTION used for asynchronous status updates
RETURNS nothing
END_PROLOG

4.3.32.2.12 DsShDescriptorProxyDCE Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.32.2.13 DsShDescriptorProxy Class

Overview:

Declarations for the DsShDescriptorProxy class. This class provides a client interface to the server's descriptor distributed object.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myHandle: void*
```

```
ourDescriptor: DsShDescriptorIDL_4_0*
```

Constructors and Destructor:

```
DsShDescriptorProxy(const RWCString& host, const RWCString&
name, const RWCString& vers);
```

normal constructor BEGIN_PROLOG

NAME DsShDescriptorProxy

DESCRIPTION Normal constructor, called via the proxy factory. Creates a proxy descriptor from the generic DCE pointer that is obtained from the server when a new server-side (real) descriptor is created.

RETURNS nothing

END_PROLOG

```
DsShDescriptorProxy(const DsShDescriptorProxy& );
```

copy constructor - NOT IMPLEMENTED

```
~DsShDescriptorProxy();
```

destructor BEGIN_PROLOG

NAME ~DsShDescriptorProxy

DESCRIPTION destructor, automatically calls Delete (via the factory) to delete the corresponding server-side object.

RETURNS nothing

END_PROLOG

Operations:

- ConstructedOK

  `EcUtStatus ConstructedOK();`

  REMOTE FUNCTIONS make sure server side is ok BEGIN_PROLOG

  NAME ConstructedOK

  DESCRIPTION This service indicates whether the server side object was created successfully.

  RETURNS Status indicating SUCCESS or FAILURE of object creation

  END_PROLOG

- GetMCF

  `EcUtStatus GetMCF(ostream& os);`

  get metadata configuration file BEGIN_PROLOG

  NAME GetMCF

  DESCRIPTION Obtains the metadata configuration file (MCF) from the server side object

  RETURNS GlStatus

  END_PROLOG

- GetQueryableParameters

  `EcUtStatus GetQueryableParameters(GlParameterList*& pl);`

  obtain queryable parameters BEGIN_PROLOG

  NAME GetQueryableParameters

  DESCRIPTION Obtains the queryable parameters for this type from the server, storing it in the given GlParameterList

  RETURNS nothing

  END_PROLOG

- operator =

  `DsShDescriptorProxy& operator =(const DsShDescriptorProxy& );`

  assignment operator - NOT IMPLEMENTED

- Validate

  `EcUtStatus Validate(istream& is, GlStatus& stat);`

  output stream Validate metadata BEGIN_PROLOG

  NAME Validate

DESCRIPTION Validates the given stream of metadata values by passing them on to the server side object which does the real validation

RETURNS GlStatus

END_PROLOG

4.3.32.2.14 DsShDescriptorRealDCE Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.32.2.15 DsShDescriptorReal Class

Overview:

Declarations for the class DsShDescriptorReal. This class provides a server interface to the server's request distributed object. It inherits from the DCE-generated server request class, and adds functions to provide stronger type-checking.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsShDescriptorReal();
```

default constructor BEGIN_PROLOG
NAME DsShDescriptorReal
DESCRIPTION
Default constructor for this abstract base class.
RETURNS nothing
GLOBALS none
FILES none
END_PROLOG

```
DsShDescriptorReal(const DsShDescriptorReal& );
```

copy constructor - NOT IMPLEMENTED

```
virtual  ~DsShDescriptorReal();
```

destructor BEGIN_PROLOG
NAME ~DsShDescriptorReal
DESCRIPTION Destructor for this class.
RETURNS nothing
GLOBALS none
FILES none
END_PROLOG

Operations:

- DistConstructedOK

```
virtual EcUtStatus DistConstructedOK() const = 0;
```

- DistGetMCF

```
virtual EcUtStatus DistGetMCF(ostream& ) const = 0;
```

DISTRIBUTED FUNCTIONS

- DistGetQueryableParameters

```
virtual EcUtStatus DistGetQueryableParameters(GlParameterList& )
const = 0;
```

- DistValidate

```
virtual EcUtStatus DistValidate(istream& , GlStatus& ) const = 0;
```

- operator =

```
DsShDescriptorReal& operator =(const DsShDescriptorReal& );
```

assignment operator - NOT IMPLEMENTED

4.3.32.2.16 DsShDOError Class

Overview:

This macro allows easier syntax in caller's throw lines, e.g.: throw DsShError("exception!", DsShErrorArgs);

(the __FILE__ and __LINE__ macros are supplied by the compiler, and will have the values of the current file name and line number where this macro is expanded) Use this macro to derive new exceptions Here are the derived exceptions Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShError

Attributes:

Constructors and Destructor:

```
DsShDOError(const char* tm, const char* fn, unsigned int ln);
```

Operations:

4.3.32.2.17 DsShDOId Class

Overview:

Declaration of a distributed object (DO) id, independent of the underlying technology. It is intended that derived classes will be created for each DO technology, and that interfaces can be designed that can refer to object IDs without being aware of the underlying technology.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myType: const enum DsShGlobal::DsEShDOType
```

Constructors and Destructor:

```
DsShDOId(enum DsShGlobal::DsEShDOType t);
```

normal constructor

```
virtual  ~DsShDOId();
```

destructor BEGIN_PROLOG

NAME ~DsShDOId

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- Equal

```
virtual int Equal(const DsShDOId& rhs) const = 0;
```

comparison BEGIN_PROLOG

NAME Equal

DESCRIPTION comparison of one DsShDOId to another

RETURNS true if they're the same, false otherwise

END_PROLOG

- operator ==

  ```
  int operator ==(const DsShDOId& rhs) const;
  ```

  easier syntax for Equal() BEGIN_PROLOG

  NAME operator=

  DESCRIPTION the comparison operator, just calls Equal()

  RETURNS boolean

  END_PROLOG

4.3.32.2.18 DsShDOInterface Class

Overview:

Declarations for the class DsShDOInterface. This class provides a basic interface that all distributed object (DO) technologies must supply for the dataserver. It is intended that derived classes will be created for each DO technology upon which dataserver(s) may be written. Derived classes are expected to provide more functions in addition to this interface) relevant to the specific technology.

Distributed Object: No BEGIN_PROLOG

NAME DsShDOInterface.h

DESCRIPTION Declarations for the class DsShDOInterface. This class provides a basic interface that all distributed object (DO) technologies must supply for the dataserver. It is intended that derived classes will be created for each DO technology upon which dataserver(s) may be written. Derived classes are expected to provide more functions in addition to this interface) relevant to the specific technology.

END_PROLOG

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsShDOInterface();
```

default constructor BEGIN_PROLOG

NAME DsShDOInterface
DESCRIPTION normal constructor
RETURNS nothing
END_PROLOG

```
virtual  ~DsShDOInterface();
```

destructor BEGIN_PROLOG
NAME ~DsShDOInterface
DESCRIPTION destructor
RETURNS nothing
END_PROLOG

Operations:

- Finish

```
virtual void Finish() = 0;
```

call to finish cleanup

- Shutdown

```
virtual void Shutdown() = 0;
```

called on server shutdown

- Start

```
virtual void Start() = 0;
```

called to start the server

4.3.32.2.19 DsShDONotFound Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:
Inherits from DsShDOError

Attributes:

Constructors and Destructor:

```
DsShDONotFound(const char* tm, const char* fn, unsigned int ln);
```

Operations:

4.3.32.2.20 DsShDORef Class

Overview:

Declarations for the class DsShDORef. This class encapsulates references to distributed objects in a manner that is independent of the underlying distributed object (DO) technology. It is intended that derived classes will be created to represent references for each DO technology upon which dataserver(s) are implemented.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myType: enum DsShGlobal::DsEShDOType
```

Constructors and Destructor:

```
DsShDORef(enum DsShGlobal::DsEShDOType t);
```

constructor

```
virtual  ~DsShDORef();
```

destructor BEGIN_PROLOG

NAME ~DsShDORef

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- Equal

  ```
  virtual int Equal(const DsShDORef& rhs) const = 0;
  ```

  comparison BEGIN_PROLOG

  NAME Equal

  DESCRIPTION compares this with the given argument

  RETURNS boolean

  END_PROLOG

- operator ==

  ```
  int operator ==(const DsShDORef& rhs) const;
  ```

  comparison operator BEGIN_PROLOG

  NAME operator==

  DESCRIPTION comparison operator, just calls Equal()

  RETURNS boolean

  END_PROLOG

4.3.32.2.21 DsShError Class

Overview:

A base class for dataserver exception types. Inherits from RWxmsg to provide an industry-standard interface.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
file: const char*
```

```
line: unsigned int
```

file name

Constructors and Destructor:

```
DsShError(const char* m, const char* f, unsigned int l);
```

normal constructor

```
virtual  ~DsShError();
```

line number since there are no allocated attributes, the default copy constructor and assignment operator are correct destructor BEGIN_PROLOG

NAME ~DsShError

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- GetFile

```
const char* GetFile() const;
```

get filename BEGIN_PROLOG

NAME GetFile

DESCRIPTION Used to obtain the file name associated with this object.

RETURNS the file name in a constant character string

END_PROLOG

- GetLine

```
const unsigned int GetLine() const;
```

get line number BEGIN_PROLOG

NAME GetLine

DESCRIPTION Used to obtain the line number associated with this object.

RETURNS the line number

END_PROLOG

- GetMsg

```
const char* GetMsg() const;
```

get text message BEGIN_PROLOG

NAME GetMsg

DESCRIPTION Used to obtain the message associated with this object.

RETURNS the text message in a constant character string

END_PROLOG

4.3.32.2.22 DsShGenCommandImp Class

Overview:

Declaration for the implementation class for commands. This class is designed to be transferred from the client process space to the server process space, and is thus derived from RWCollectable, so that it can be "flattened" and "restored". This is the core data of every command, around which a more friendly and useful interface is provided by higher-level client-side and server-side classes.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myParameters: GlParameterList`

name of service

`myServiceName: RWCString`

Constructors and Destructor:

`DsShGenCommandImp(const DsShGenCommandImp& rhs);`

parameters for service copy constructor BEGIN_PROLOG

NAME DsShGenCommandImp

DESCRIPTION copy constructor. RWCStrings are shallow/reference-counted, GlParameterLists are shallow.

RETURNS nothing

END_PROLOG

`DsShGenCommandImp(const RWCString& svc, const GlParameterList& pl);`

normal constructor BEGIN_PROLOG

NAME DsShGenCommandImp

DESCRIPTION normal constructor. Initializes the service name, parameters, and category.

RETURNS nothing

END_PROLOG

```
DsShGenCommandImp();
```

default constructor

```
virtual  ~DsShGenCommandImp();
```

destructor BEGIN_PROLOG

NAME ~DsShGenCommandImp

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- binaryStoreSize

```
virtual unsigned long binaryStoreSize() const;
```

RWCollectable function to obtain binary size of object BEGIN_PROLOG

NAME binaryStoreSize

DESCRIPTION a required RWCollectable function that calculates the binary size of an object

RETURNS the size (in bytes) of this object

END_PROLOG

- copy

```
virtual RWCollectable* copy() const;
```

- GetParameters

```
GlParameterList& GetParameters();
```

- GetParameters

```
const GlParameterList& GetParameters() const;
```

get the current parameters BEGIN_PROLOG

NAME GetParameters

DESCRIPTION obtain the current parameters for this command.

RETURNS constant reference to parameters

END_PROLOG

- GetServiceName

```
const RWCString& GetServiceName() const;
```

get the current service name BEGIN_PROLOG

NAME GetServiceName

DESCRIPTION returns the current service name for this command

RETURNS constant string reference containing service name

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

macro instantiation required for all RWCollectable classes

- operator ==

```
virtual int operator ==(const DsShGenCommandImp& ) const;
```

equality operator BEGIN_PROLOG

NAME operator==

DESCRIPTION equality operator. Required for classes that are stored in RW collections, and DsShGenCommandImp is stored in a RW collection in DsShGenRequestImp.

RETURNS boolean

END_PROLOG

- operator =

```
DsShGenCommandImp& operator =(const DsShGenCommandImp& );
```

assignment operator

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS nothing

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS nothing

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

RWCollectable functions to save and restore objects to a stream BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

RETURNS nothing

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS nothing

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

produces human-readable output BEGIN_PROLOG

NAME saveOn

DESCRIPTION produces a human-readable description of this object

RETURNS nothing

END_PROLOG

- SetParameters

```
void SetParameters(const GlParameterList& pl);
```

set the parameters BEGIN_PROLOG

NAME SetParameters

DESCRIPTION Set the parameters for this command.

RETURNS nothing

END_PROLOG

- SetServiceName

  `void SetServiceName(const RWCString& svc);`

  set the service name BEGIN_PROLOG

  NAME SetServiceName

  DESCRIPTION set the service name for this command

  RETURNS nothing

  END_PROLOG

4.3.32.2.23 DsShGenRequestImp Class

Overview:

Declaration for the implementation class for requests. This class is designed to be transferred from the client process space to the server process space, and is thus derived from RWCollectable, so that it can be "flattened" and "restored". This is the core data of every command, around which a more friendly and useful interface is provided by higher-level client-side and server-side classes.

Distributed Object:No

Export Control: Public

Inheritance Relationships:

Attributes:

`myCommands: RWOrdered`

type of server for this request

`myServerType: RWCString`

Constructors and Destructor:

`DsShGenRequestImp(const DsShGenRequestImp& rhs);`

type of server copy constructor BEGIN_PROLOG

NAME DsShGenRequestImp

DESCRIPTION copy constructor. RWCStrings are shallow/reference-counted, domain and commands are shallow.

RETURNS nothing

END_PROLOG

`DsShGenRequestImp(const RWCString& stype);`

normal constructor

`DsShGenRequestImp();`

default constructor, required by RWCollectable

`virtual  ~DsShGenRequestImp();`

request to assign destructor BEGIN_PROLOG

NAME ~DsShGenRequestImp

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- binaryStoreSize

  `virtual unsigned long binaryStoreSize() const;`

  RWCollectable function to obtain binary size of object BEGIN_PROLOG

  NAME binaryStoreSize

  DESCRIPTION a required RWCollectable function that calculates the binary size of an object

  RETURNS the size (in bytes) of this object

  END_PROLOG

- Commands

  `RWOrdered* Commands();`

  server type string provides access to the commands BEGIN_PROLOG

  NAME Commands

  DESCRIPTION provides access to the underlying command vector

  RETURNS pointer to the base class of the vector of commands

  END_PROLOG

- copy

  `virtual RWCollectable* copy() const;`

- EstimateXDRSize

  `virtual long EstimateXDRSize() const;`

estimates size of buffer needed to hold XDR'd request BEGIN_PROLOG

NAME EstimateXDRSize

DESCRIPTION Returns the number of bytes that should be allocated for a buffer that is expected to hold the XDR data for the current contents of this request. As in GlParameterList, the exact size cannot be known until the request is actually flattened, but this function returns a number always larger than necessary to provide a satisfactory margin for error. When the request is flattened, its actual XDR size is returned.

RETURNS estimated XDR buffer size

END_PROLOG

- Flatten

```
virtual long Flatten(char* buffer, long buflen) const;
```

save the request in architecture independent XDR format into a user-supplied buffer BEGIN_PROLOG

NAME Flatten

DESCRIPTION stores a request in architecture-independent format in the given buffer

RETURNS the count of bytes actually used in the buffer

GLOBALS none

FILES none

END_PROLOG

- Flatten

```
virtual void Flatten(const RWCString& fname) const;
```

any output stream save the request in architecture independent XDR format into a file BEGIN_PROLOG

NAME Flatten

DESCRIPTION stores a request in architecture-independent format in a file with the given name

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- GetServerType

```
const RWCString& GetServerType() const;
```

get the server type BEGIN_PROLOG

NAME GetServerType

DESCRIPTION returns the server type

RETURNS nothing

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

macro instantiation required for all RWCollectable classes

- operator ==

```
int operator ==(const DsShGenRequestImp& ) const;
```

equality operator BEGIN_PROLOG

NAME operator==

DESCRIPTION equality operator

RETURNS boolean

END_PROLOG

- operator =

```
DsShGenRequestImp& operator =(const DsShGenRequestImp& );
```

assignment operator

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS nothing

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS nothing

END_PROLOG

- Restore

  `static long Restore(char* buffer, long buflen, DsShGenRequestImp*& newobj);`

  restore a request from a XDR format buffer BEGIN_PROLOG

  NAME Restore

  DESCRIPTION load a request from an architecture-independent buffer

  RETURNS the number of bytes actually used from the buffer

  GLOBALS none

  FILES none

  END_PROLOG

- Restore

  `static void Restore(const RWCString& fname, DsShGenRequestImp*& newobj);`

  restore a request from a XDR format file BEGIN_PROLOG

  NAME Restore

  DESCRIPTION load a request from an architecture-independent file with the given name

  RETURNS nothing

  GLOBALS none

  FILES none

  END_PROLOG

- saveGuts

  `virtual void saveGuts(RWvostream& vos) const;`

  request to compare RWCollectable functions to save and restore objects to a stream BEGIN_PROLOG

  NAME saveGuts

  DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

  RETURNS nothing

  END_PROLOG

- saveGuts

  `virtual void saveGuts(RWFile& f) const;`

  BEGIN_PROLOG

  NAME saveGuts

  DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS nothing
END_PROLOG

- saveOn

  `virtual void saveOn(ostream& os) const;`
  produces human-readable output BEGIN_PROLOG
  NAME saveOn
  DESCRIPTION produces a human-readable description of this object
  RETURNS nothing
  END_PROLOG

- SetServerType

  `void SetServerType(const RWCString& type);`
  set the server type BEGIN_PROLOG
  NAME SetServerType
  DESCRIPTION sets the server type
  RETURNS nothing
  END_PROLOG

4.3.32.2.24 DsShGlobal Class

Overview:
global declarations/constants for sh Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.32.2.25 DsShMutexLock Class

Overview:

Declarations for the class DsShMutexLock. This class provides a safe method of locking and unlocking DsShMutex objects, especially in the presence of exceptions. Using this class, programmer errors are reduced since the programmer is not require to remember to unlock the mutex (the destructor does it automatically).

Distributed Object: No

Export Control: Public

Inheritance Relationships:

Attributes:

```
myMutex: DsShMutex&
```

Constructors and Destructor:

```
DsShMutexLock(DsShMutex& m);
```

constructor - locks the mutex

```
DsShMutexLock(const DsShMutexLock& );
```

copy constructor NOT IMPLEMENTED

```
~DsShMutexLock();
```

destructor - unlocks the mutex BEGIN_PROLOG

NAME ~DsShMutexLock

DESCRIPTION destructor, unlocks the mutex

RETURNS nothing

END_PROLOG

Operations:

- operator =

```
DsShMutexLock& operator =(const DsShMutexLock& );
```

assignment operator NOT IMPLEMENTED

4.3.32.2.26 DsShMutexValVector ParameterizedClass

Overview:

Declarations for the class DsShMutexValVector. This template makes it easy to create a RWTValOrderedVector with an associated DsShMutex (one of its superclasses). This saves the awkwardness of creating a separately named mutex and prevents potential programmer errors (such as locking/unlocking the wrong mutex).

NOTES Locking and unlocking are safely performed by using a DsShMutexLock object together with a DsShMutexValVector object. (See comments in DsShMutexLock.h)

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShMutex

Attributes:

Constructors and Destructor:

```
DsShMutexValVector(unsigned int capac);
```

constructor

```
DsShMutexValVector(const DsShMutexValVector& rhs);
```

copy constructor

```
virtual  ~DsShMutexValVector();
```

destructor destructor

Operations:

- operator =

```
DsShMutexValVector& operator =(const DsShMutexValVector& rhs);
```

assignment operator assignment operator

4.3.32.2.27 DsShMutex Class

Overview:

Declarations for the class DsShMutex. This class provides a mutex class that encapsulates the underlying (currently DCE) mutex implementation. The underlying implementation can be replaced without affecting users of this class.

NOTES Locking and unlocking are safely performed by using a DsShMutexLock object together with a DsShMutexLock object. (See comments in DsShMutexLock.h)

Declarations for the class DsShMutexLock. This class provides a safe method of locking and unlocking DsShMutex objects, especially in the presence of exceptions. Using this class, programmer errors are reduced since the programmer is not require to remember to unlock the mutex (the destructor does it automatically).

Declarations for the class DsShMutexValVector. This template makes it easy to create a RWTValOrderedVector with an associated DsShMutex (one of its superclasses). This saves the awkwardness of creating a separately named mutex and prevents potential programmer errors (such as locking/unlocking the wrong mutex).

NOTES Locking and unlocking are safely performed by using a DsShMutexLock object together with a DsShMutexValVector object. (See comments in DsShMutexLock.h)

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myDCEMutex: DCEPthreadMutex*
```

Constructors and Destructor:

```
DsShMutex();
```

constructor BEGIN_PROLOG

NAME DsShMutex

DESCRIPTION constructor

RETURNS nothing

END_PROLOG

```
DsShMutex(const DsShMutex& );
```

copy constructor NOT IMPLEMENTED

```
~DsShMutex();
```

destructor BEGIN_PROLOG

NAME ~DsShMutex

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- Lock

```
void Lock();
```

lock the mutex, sleep trying BEGIN_PROLOG

NAME Lock

DESCRIPTION Locks the mutex. If the mutex is already locked, sleeps (FIFO) until it's available, then locks it.

RETURNS nothing

END_PROLOG

- operator =

```
DsShMutex& operator =(const DsShMutex& );
```

assignment operator NOT IMPLEMENTED

- TryLock

```
int TryLock();
```

try to lock the mutex, but don't sleep BEGIN_PROLOG

NAME TryLock

DESCRIPTION Attempts to lock the mutex. If the mutex is already locked, returns FALSE immediately. Otherwise locks the mutex and returns TRUE.

RETURNS boolean

END_PROLOG

- Unlock

```
void Unlock();
```

unlock the mutex BEGIN_PROLOG

NAME Unlock

DESCRIPTION Unlocks the mutex.

RETURNS nothing

END_PROLOG

4.3.32.2.28 DsShQuitProxy Class

Overview:

Declaration of the DsShQuitProxy class, which provides a client interface to the server shutdown class. This class will no longer be needed when the MSS lifecycle services and/or the managed process framework are complete.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myQuit: DsShQuitIDL_4_0*
```

Constructors and Destructor:

```
DsShQuitProxy(char* netaddr);
```

normal constructor

```
DsShQuitProxy();
```

hidden default constructor

```
~DsShQuitProxy();
```

network address to connect, e.g., hostname destructor

Operations:

- Shutdown

```
void Shutdown();
```

the only method: shutdown the server

4.3.32.2.29 DsShQuitReal Class

Overview:

Declaration of the DsShQuitReal class. This is a server-side class that implements code to shutdown the dataserver. This class will no longer be needed when the MSS lifecycle services and/or the managed process framework are complete.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsShQuitReal();
```

normal constructor

```
~DsShQuitReal();
```

destructor BEGIN_PROLOG

NAME ~DsShQuitReal

DESCRIPTION destructor

RETURNS nothing

END_PROLOG

Operations:

- AnotherServer

```
static int AnotherServer();
```

check for other server BEGIN_PROLOG

NAME AnotherServer

DESCRIPTION Checks if another server is running on the current host.

RETURNS TRUE if there is another server, FALSE otherwise.

END_PROLOG

- CrashCleanup

```
static void CrashCleanup();
```

used to cleanup after a crash BEGIN_PROLOG

NAME CrashCleanup

DESCRIPTION The function that is actually called from the signal handlers, performs all graceful cleanup activities (including DCE cleanup)

RETURNS nothing

END_PROLOG

- CrashClear

```
static void CrashClear();
```

used to clear signal handlers after a crash BEGIN_PROLOG

NAME CrashClear

DESCRIPTION Used to clear the signal handlers when a crash occurs, to prevent recursive calls (e.g., another segment fault (or whatever) could occur while trying to clean up from one!)

RETURNS nothing

END_PROLOG

- CrashHandler

```
static void CrashHandler();
```

static function used to handle the common crash signals: BUS, SEGV, ILL BEGIN_PROLOG

NAME CrashHandler

DESCRIPTION Handles the crash signals, BUS: e.g., use of corrupt pointer SEGV: e.g., use of a null pointer ILL: illegal instruction, e.g., corrupted stack on function return

RETURNS nothing

END_PROLOG

- CrashSetup

```
void CrashSetup();
```

LOCAL FUNCTIONS setup all signal handling, etc., for crash handling and cleanup BEGIN_PROLOG

NAME CrashSetup

DESCRIPTION function used by the server to setup signal handlers to catch all of the programming error signals to allow graceful shutdown.

RETURNS nothing

END_PROLOG

- Shutdown

```
void Shutdown();
```

DISTRIBUTED FUNCTIONS shutdown the server BEGIN_PROLOG

NAME Shutdown

DESCRIPTION The distributed shutdown function, tells the factory to clean up, then exits via another thread in an attempt to let DCE return to the client who called Shutdown. This doesn't always work, and the client will hang for about 40 seconds before getting a DCE exception.

RETURNS nothing

END_PROLOG

4.3.32.2.30 DsShSciCommandImp Class

Overview:

Declarations for the class DsShSciCommandImp. This class specializes DsShGenCommandImp for the science data server to provide command categories.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShGenCommandImp

Attributes:

`myCategory: enum DsShGlobal::DsEShSciCommandCategory`

for access to myCategory, etc.

Constructors and Destructor:

`DsShSciCommandImp(const DsShSciCommandImp& rhs);`

category of command copy constructor

`DsShSciCommandImp(const RWCString& svc, const GlParameterList& pl, enum DsShGlobal::DsEShSciCommandCategory cat);`

normal constructor

`DsShSciCommandImp();`

default constructor

`virtual ~DsShSciCommandImp();`

destructor BEGIN_PROLOG

NAME ~DsShSciCommandImp

DESCRIPTION destructor

RETURNS nothing
END_PROLOG

Operations:

- binaryStoreSize

  ```
  unsigned long binaryStoreSize() const;
  ```

  RWCollectable function to obtain binary size of object BEGIN_PROLOG
  NAME binaryStoreSize
  DESCRIPTION a required RWCollectable function that calculates the binary size of an object
  RETURNS the size (in bytes) of this object
  END_PROLOG

- copy

  ```
  virtual RWCollectable* copy() const;
  ```

- GetCategory

  ```
  enum DsShGlobal::DsEShSciCommandCategory GetCategory() const;
  ```

  get the current category BEGIN_PROLOG
  NAME GetCategory
  DESCRIPTION get the current category for this command
  RETURNS the category
  END_PROLOG

- isA

  ```
  virtual unsigned short isA() const;
  ```

- newSpecies

  ```
  virtual RWCollectable* newSpecies() const;
  ```

  macro instantiation required for all RWCollectable classes

- operator =

  ```
  DsShSciCommandImp& operator =(const DsShSciCommandImp& );
  ```

  assignment operator

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS nothing

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS nothing

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

RWCollectable functions to save and restore objects to a stream BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

RETURNS nothing

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS nothing

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

produces human-readable output BEGIN_PROLOG

NAME saveOn

DESCRIPTION produces a human-readable description of this object

RETURNS nothing

END_PROLOG

- SetCategory

```
void SetCategory(enum DsShGlobal::DsEShSciCommandCategory cat);
```

set the category BEGIN_PROLOG

NAME SetCategory

DESCRIPTION set the category for the command

RETURNS nothing

END_PROLOG

4.3.32.2.31 DsShSciRequestImp Class

Overview:

Declarations for the DsShSciRequestImp class. This class specializes DsShGenRequestImp for the science server, adding priorities and domains.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShGenRequestImp

Attributes:

```
myDomain: RWTPtrOrderedVector<class GlUR>
```

priority of this request

```
myPriority: enum DsShGlobal::DsEShSciPriority
```

Constructors and Destructor:

```
DsShSciRequestImp();
```

default constructor

```
DsShSciRequestImp(const DsShSciRequestImp& rhs);
```

copy constructor

```
virtual  ~DsShSciRequestImp();
```

destructor BEGIN_PROLOG
NAME ~DsShSciRequestImp
DESCRIPTION destructor
RETURNS nothing
GLOBALS none
FILES none
END_PROLOG

Operations:

- apply

```
void apply(void (*) (class RWCollectable*, void*) ap, void* x);
```

- at

```
DsShSciCommandImp*& at(unsigned int i);
```

- at

```
const DsShSciCommandImp* at(unsigned int i) const;
```

- binaryStoreSize

```
virtual unsigned long binaryStoreSize() const;
```

RWCollectable function to obtain binary size of object BEGIN_PROLOG
NAME binaryStoreSize
DESCRIPTION a required RWCollectable function that calculates the binary size of an object
RETURNS the size (in bytes) of this object
GLOBALS none
FILES none
END_PROLOG

- clearAndDestroy

```
void clearAndDestroy();
```

- clear

```
void clear();
```

- contains

```
int contains(const DsShSciCommandImp* p) const;
```

- copy

```
virtual RWCollectable* copy() const;
```

- entries

```
unsigned int entries() const;
```

- find

```
DsShSciCommandImp* find(const DsShSciCommandImp* p) const;
```

- first

```
DsShSciCommandImp* first() const;
```

- GetDomain

```
const RWTPtrOrderedVector<class GlUR>& GetDomain() const;
```

obtain the current domain for the request BEGIN_PROLOG
NAME GetDomain
DESCRIPTION obtains the current domain of the request
RETURNS the domain
END_PROLOG

- GetPriority

```
enum DsShGlobal::DsEShSciPriority GetPriority() const;
```

obtain the current priority of the request BEGIN_PROLOG
NAME GetPriority
DESCRIPTION obtains the priority of the request
RETURNS the priority
END_PROLOG

- index

```
unsigned int index(const DsShSciCommandImp* p) const;
```

- insertAt

```
void insertAt(unsigned int i, DsShSciCommandImp* p);
```

- insert

```
DsShSciCommandImp* insert(DsShSciCommandImp* p);
```

- isA

```
virtual unsigned short isA() const;
```

- isEmpty

```
int isEmpty() const;
```

- last

```
DsShSciCommandImp* last() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

macro instantiation required for all RWCollectable classes

- occurrencesOf

```
unsigned int occurrencesOf(const DsShSciCommandImp* p) const;
```

- operator =

```
DsShSciRequestImp& operator =(const DsShSciRequestImp& );
```

assignment operator

- operator []

```
DsShSciCommandImp*& operator [](unsigned int i);
```

- prepend

```
DsShSciCommandImp* prepend(DsShSciCommandImp* p);
```

- removeAndDestroy

```
void removeAndDestroy(const DsShSciCommandImp* p);
```

- remove

```
DsShSciCommandImp* remove(const DsShSciCommandImp* p);
```

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

RWCollectable functions to save and restore objects to a stream BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
virtual void saveOn(ostream& os) const;
```

produces human-readable output BEGIN_PROLOG

NAME saveOn

DESCRIPTION produces a human-readable description of this object

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- SetDomain

```
void SetDomain(RWTPtrOrderedVector<class GlUR>& domain);
```

set the domain for the request BEGIN_PROLOG

NAME SetDomain

DESCRIPTION set the domain for this request

RETURNS nothing

END_PROLOG

- SetPriority

```
void SetPriority(enum DsShGlobal::DsEShSciPriority pty);
```

set the priority for the request BEGIN_PROLOG

NAME SetPriority

DESCRIPTION sets the priority of the request

RETURNS nothing

END_PROLOG

4.3.32.2.32 DsShSRequestProxy Class

Overview:

Declarations for the DsShSRequestProxy class. This class provides a client interface to the server's request distributed object.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myHandle: void*`

`myUpdateable: const DsShUpdateable*`

context handle

`myUpdater: DsShDCEUpdaterConcrete*`

updateable object (if any)

`ourSRequest: DsShSRequestIDL_4_0*`

the updater for the updateable

Constructors and Destructor:

`DsShSRequestProxy(const RWCString& host, DsShUpdateable* update);`

default constructor

`~DsShSRequestProxy();`

destructor BEGIN_PROLOG

NAME ~DsShSRequestProxy

DESCRIPTION destructor, automatically calls Delete (via the factory) to delete the corresponding server-side request object.

RETURNS nothing

END_PROLOG

Operations:

- GetHandle

  ```
  void* GetHandle() const;
  ```

  return the context handle

- GetID

  ```
  unsigned long GetID() const;
  ```

  get the ID of this request BEGIN_PROLOG

  NAME GetID

  DESCRIPTION Obtains the (server, not DCE) id of the request from the server.

  RETURNS the id

  END_PROLOG

- GetResults

  ```
  void GetResults(GlParameterList*& pl);
  ```

  get the current results BEGIN_PROLOG

  NAME GetResults

  DESCRIPTION Obtains the current results list from the server, storing it in the given GlParameterList

  RETURNS nothing

  END_PROLOG

- GetStatus

  ```
  void GetStatus(GlStatus& status);
  ```

  get the current status BEGIN_PROLOG

  NAME GetStatus

  DESCRIPTION Obtains the current status from the server, storing it in the given status object

  RETURNS nothing

  END_PROLOG

- Report

  ```
  void Report();
  ```

REMOTE FUNCTIONS log a message on the server identifying this request BEGIN_PROLOG

NAME Report

DESCRIPTION Causes an identifying message to be logged on the server.

RETURNS nothing

END_PROLOG

- SetImp

```
void SetImp(DsShGenRequestImp& thereq);
```

pass request core information to the server-side request BEGIN_PROLOG

NAME SetImp

DESCRIPTION Transfers the entire request (flattens) from the client to the server. Since we can't find out the size of the XDR data for the request until we actually flatten it, we use the EstimateXDRSize() function which returns an approximation that is "guaranteed" to be larger than the actual value to allocate a buffer. Once flattened, we know the actual size and transfer only those bytes to the server.

RETURNS nothing

END_PROLOG

- SetPriority

```
SetPriority();
```

4.3.32.2.33 DsShSRequestReal Class

Overview:

Declarations for the class DsShSRequestReal. This class provides a server interface to the server's request distributed object. It inherits from the DCE-generated server request class, and adds functions to provide stronger type-checking.

inheritance Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myID: unsigned long
```

core request information

myRB: DsShGenRequestImp*

current status

myResults: GlParameterList

myStatus: GlStatus

current results

myUpdater: DsShUpdaterProxy*

request's ID

ourCreated: unsigned int

running count for ID generation

ourDestroyed: unsigned int

count of requests created

ourID: unsigned long

updater for this request, if it's async

Constructors and Destructor:

DsShSRequestReal(DsShUpdaterProxy* up);

default constructor

~DsShSRequestReal();

destructor BEGIN_PROLOG

NAME ~DsShSRequestReal

DESCRIPTION destructor - due to anomalies in DCE, it is possible that a request may be deleted before it's been submitted (this happens once in several hundred requests). This problem will be eliminated in a later phase, but for phase one we simply log this event and prevent the request from submission by removing it from the internal SR vector.

RETURNS nothing

END_PROLOG

Operations:

- DistGetID

unsigned long DistGetID() const;

return the request's ID BEGIN_PROLOG

NAME GetID

DESCRIPTION returns the dataserver (not DCE) id number for this request

RETURNS

END_PROLOG

- DistGetResults

```
const GlParameterList& DistGetResults() const;
```

return the current value of the results BEGIN_PROLOG

NAME GetResults

DESCRIPTION Flattens the current value of the results list and returns it to the client.

RETURNS nothing

END_PROLOG

- DistGetStatus

```
const GlStatus& DistGetStatus() const;
```

return the current status BEGIN_PROLOG

NAME GetStatus

DESCRIPTION Flattens the current value of the status and returns it to the client.

RETURNS nothing

END_PROLOG

- DistReport

```
void DistReport() const;
```

DISTRIBUTED FUNCTIONS log a message on the server identifying this request BEGIN_PROLOG

NAME Report

DESCRIPTION logs an identifying message on the server

RETURNS nothing

END_PROLOG

- DistSetImp

```
void DistSetImp(const char* thereq, long len);
```

passes the core request information from the client to me BEGIN_PROLOG

NAME SetImp

DESCRIPTION Takes a flattened DsShGenRequestImp object, and reconstitutes it. Used by clients to transfer the "core" information of a request to the server before submitting the request for execution

RETURNS nothing

END_PROLOG

- RequestImp

  RequestImp();

  get a reference to the core request information

- Results

  GlParameterList& Results();

  get a reference to the results

- SetPriority

  SetPriority();

- Statistics

  static void Statistics();

  static function that dumps the statistics for all request to the log BEGIN_PROLOG

  NAME Statistics

  DESCRIPTION Dumps statistics about requests created and destroyed. Generally used when server is shutting down.

  RETURNS nothing

  END_PROLOG

- Status

  Status();

  get a reference to the status

- Update

  void Update() const;

  LOCAL FUNCTIONS if there's an updater, trigger it BEGIN_PROLOG

  NAME Update

  DESCRIPTION If this request currently has an updater, triggers it

  RETURNS nothing

  END_PROLOG

4.3.32.2.34 DsShSRequest_X_Y Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

- CreateBase

```
CreateBase();
```

- GetFinishTime

```
GetFinishTime();
```

- GetID

```
GetID();
```

- GetResults

```
GetResults();
```

- GetStaus

```
GetStaus();
```

- GetSubmitTime

```
GetSubmitTime();
```

- Report

```
Report();
```

- SetImp

  ```
  SetImp();
  ```

- SetUpdater

  ```
  SetUpdater();
  ```

4.3.32.2.35 DsShTypeID Class

Overview:

Declarations for the class DsShTypeID. This class represents an identifier for a science data type, encapsulating the name, version, and a code number for each type.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myCode: unsigned long
```

```
myName: RWCString
```

code

```
myVersion: RWCString
```

name of type

Constructors and Destructor:

```
DsShTypeID(unsigned long code, const RWCString& name, const
RWCString& version);
```

construct with code, name, and version

```
DsShTypeID(const DsShTypeID& rhs);
```

copy constructor

```
DsShTypeID(const RWCString& name);
```
construct with name

```
DsShTypeID(unsigned long code);
```
construct with code

```
DsShTypeID();
```
default constructor

```
DsShTypeID(const RWCString& name, const RWCString& version);
```
construct with name and version

```
~DsShTypeID();
```
destructor BEGIN_PROLOG
NAME ~DsShTypeID
DESCRIPTION destructor
RETURNS nothing
END_PROLOG

Operations:

- GetTypeCode

```
const unsigned long GetTypeCode() const;
```
obtain code BEGIN_PROLOG
NAME GetTypeCode
DESCRIPTION returns the code for this type
RETURNS the code
END_PROLOG

- GetTypeName

```
const RWCString& GetTypeName() const;
```
obtain type name BEGIN_PROLOG
NAME GetTypeName
DESCRIPTION returns the name for this type
RETURNS the name
END_PROLOG

- GetTypeVersion

```
const RWCString& GetTypeVersion() const;
```

obtain version BEGIN_PROLOG

NAME GetTypeVersion

DESCRIPTION returns the version for this type

RETURNS the version

END_PROLOG

- operator ==

```
int operator ==(const DsShTypeID& rhs) const;
```

equality operator BEGIN_PROLOG

NAME operator==

DESCRIPTION equality operator, attempts to check by codes (if set) otherwise checks names & versions

RETURNS boolean

END_PROLOG

- operator =

```
DsShTypeID& operator =(const DsShTypeID& id);
```

assignment operator BEGIN_PROLOG

NAME operator=

DESCRIPTION assignment operator - copies attributes

RETURNS reference to this object

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

produce human-readable output BEGIN_PROLOG

NAME saveOn

DESCRIPTION used to produce human-readable output of the value of an object of this class

RETURNS nothing

END_PROLOG

- SetTypeCode

```
void SetTypeCode(unsigned long code);
```

set a new code BEGIN_PROLOG

NAME SetTypeCode

DESCRIPTION sets a new type code for this type

RETURNS nothing

END_PROLOG

- SetTypeName

  `void SetTypeName(const RWCString& name);`

  set a new name BEGIN_PROLOG

  NAME SetTypeName

  DESCRIPTION sets a new type name for this type

  RETURNS nothing

  END_PROLOG

- SetTypeVersion

  `void SetTypeVersion(const RWCString& version);`

  set a new version BEGIN_PROLOG

  NAME SetTypeVersion

  DESCRIPTION sets a new version for this type

  RETURNS nothing

  END_PROLOG

- Textify

  `RWCString Textify() const;`

  provide a string representation of this type BEGIN_PROLOG

  NAME Textify

  DESCRIPTION Create an ascii representation of the datatype id

  RETURNS RWCString

  END_PROLOG

4.3.32.2.36 DsShUpdateable Class

Overview:

Declartions for the DsShUpdateable class. This class provides a interface for any class that wants to provide an asynchronous update interface.

BEGIN_PROLOG

NAME DsShUpdateable.h

DESCRIPTION Declartions for the DsShUpdateable class. This class provides a interface for any class that wants to provide an asynchronous update interface.

END_PROLOG Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
virtual  ~DsShUpdateable();
```

destructor

Operations:

- AsyncUpdate

```
virtual void AsyncUpdate() = 0;
```

called on every async update

4.3.32.2.37 DsShUpdaterProxy Class

Overview:

Declarations for the DsShUpdaterProxy class. This class provides a client interface to the server's Updater distributed object.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myUpdater: DsShUpdaterIDL_4_0*
```

Constructors and Destructor:

```
DsShUpdaterProxy(const DsShDCERef& ref);
```

normal constructor BEGIN_PROLOG

NAME DsShUpdaterProxy

DESCRIPTION normal constructor.

RETURNS nothing

END_PROLOG

```
DsShUpdaterProxy(const DsShUpdaterProxy& );
```

prevent copying

```
~DsShUpdaterProxy();
```

destructor BEGIN_PROLOG

NAME ~DsShUpdaterProxy

DESCRIPTION destructor. Destroys the underlying DCE object and calls the Delete function (via the factory) to delete my server-side component.

RETURNS nothing

END_PROLOG

Operations:

- operator =

```
DsShUpdaterProxy operator =(const DsShUpdaterProxy& );
```

- Update

```
void Update();
```

perform the update BEGIN_PROLOG

NAME Update

DESCRIPTION performs the update

RETURNS nothing

END_PROLOG

4.3.32.2.38 DsShUpdaterReal Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShUpdater_X_Y_ABS

Attributes:

```
myCallback:

myLastStatus: GlStatus*:
```

Constructors and Destructor:

```
DsShUpdaterReal();
```

Operations:

- DistReport

```
DistReport();
```

- DistUpdate

```
DistUpdate();
```

- GetLastStatus

```
GetLastStatus();
```

4.3.32.2.39 DsShUpdater_X_Y_ABS Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

- Report

```
Report();
```

- UpdateStatus

```
UpdateStatus();
```

- Update

```
Update();
```

4.3.32.2.40 DsShUpdater_X_Y Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

- Report

```
Report();
```

- UpdateStatus

```
UpdateStatus();
```

- Update

```
Update();
```

4.3.33 sr Class Category

4.3.33.1 Overview

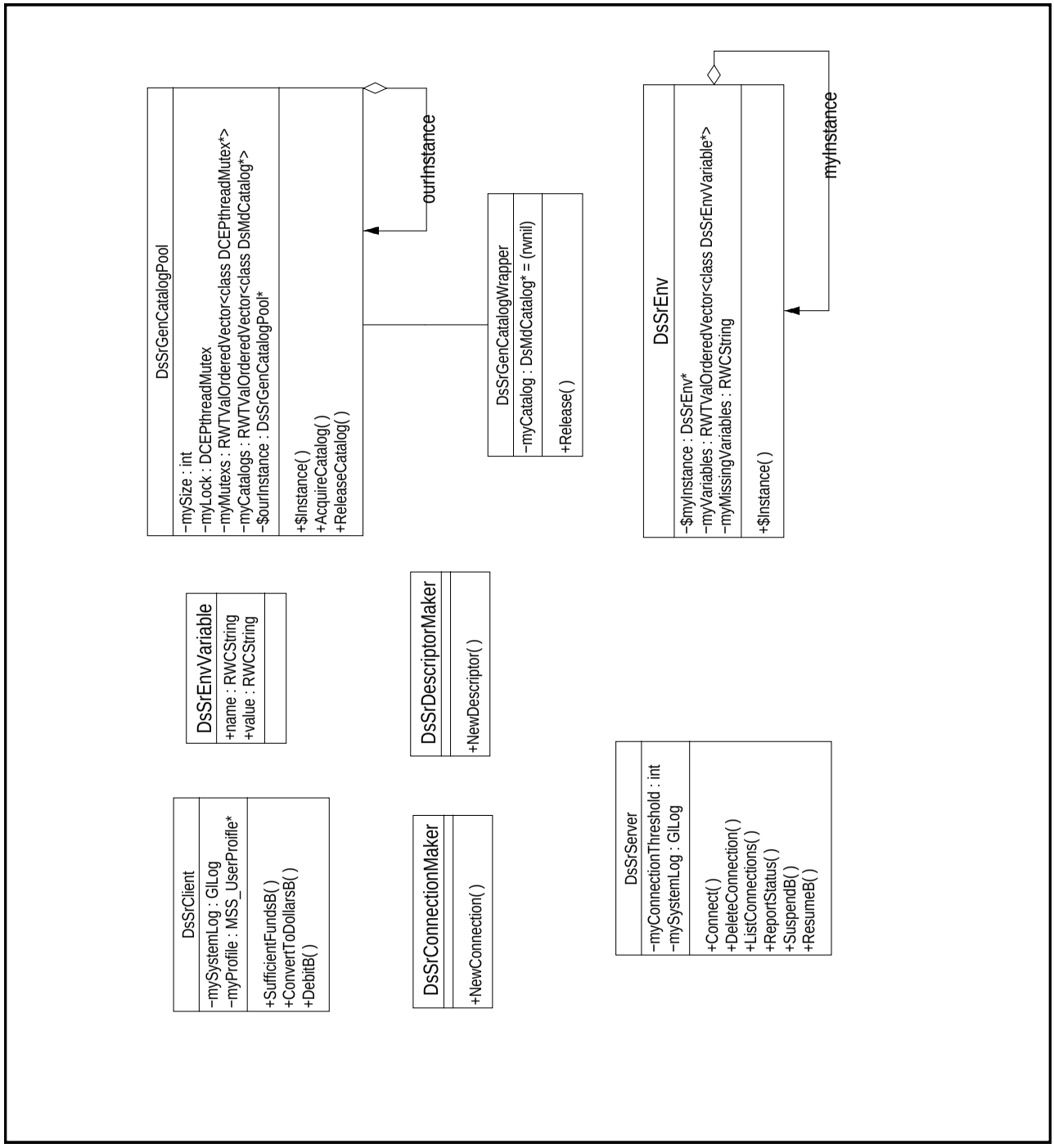


Figure 52 DsSr1

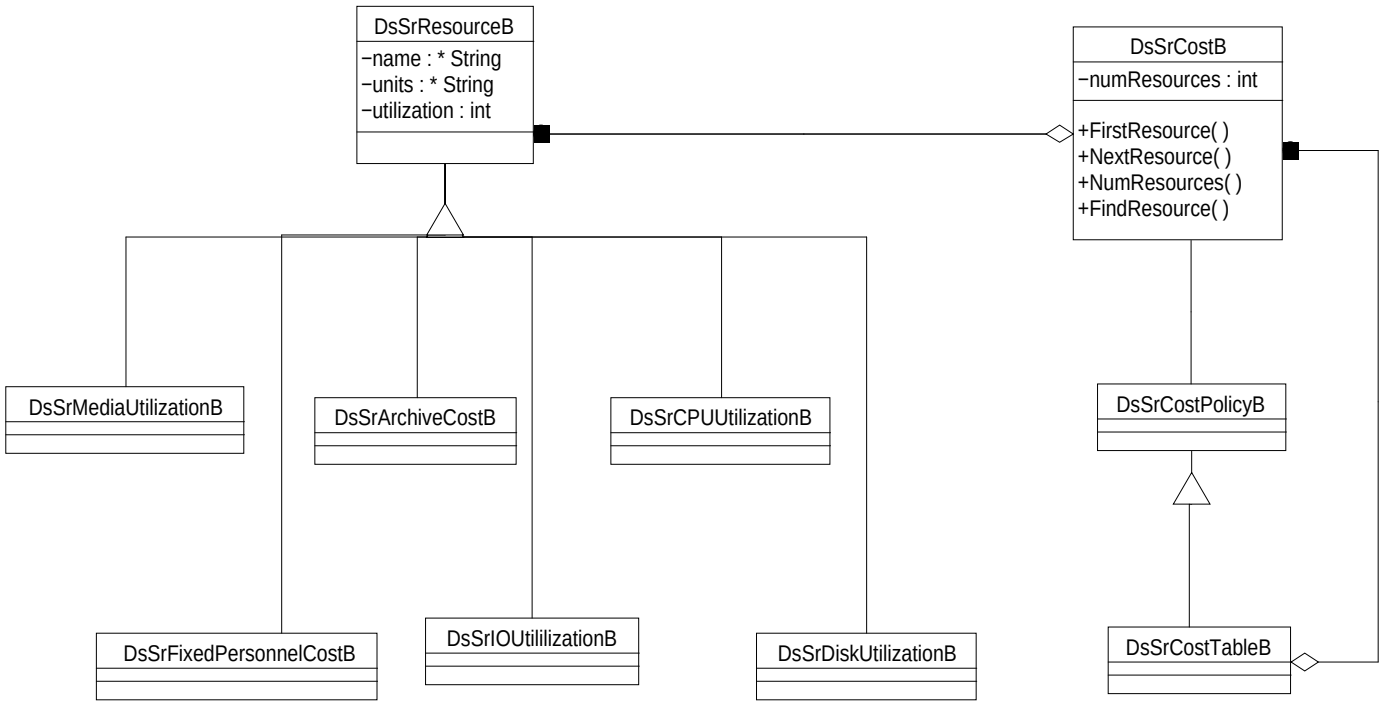


Figure 53 DsSr2

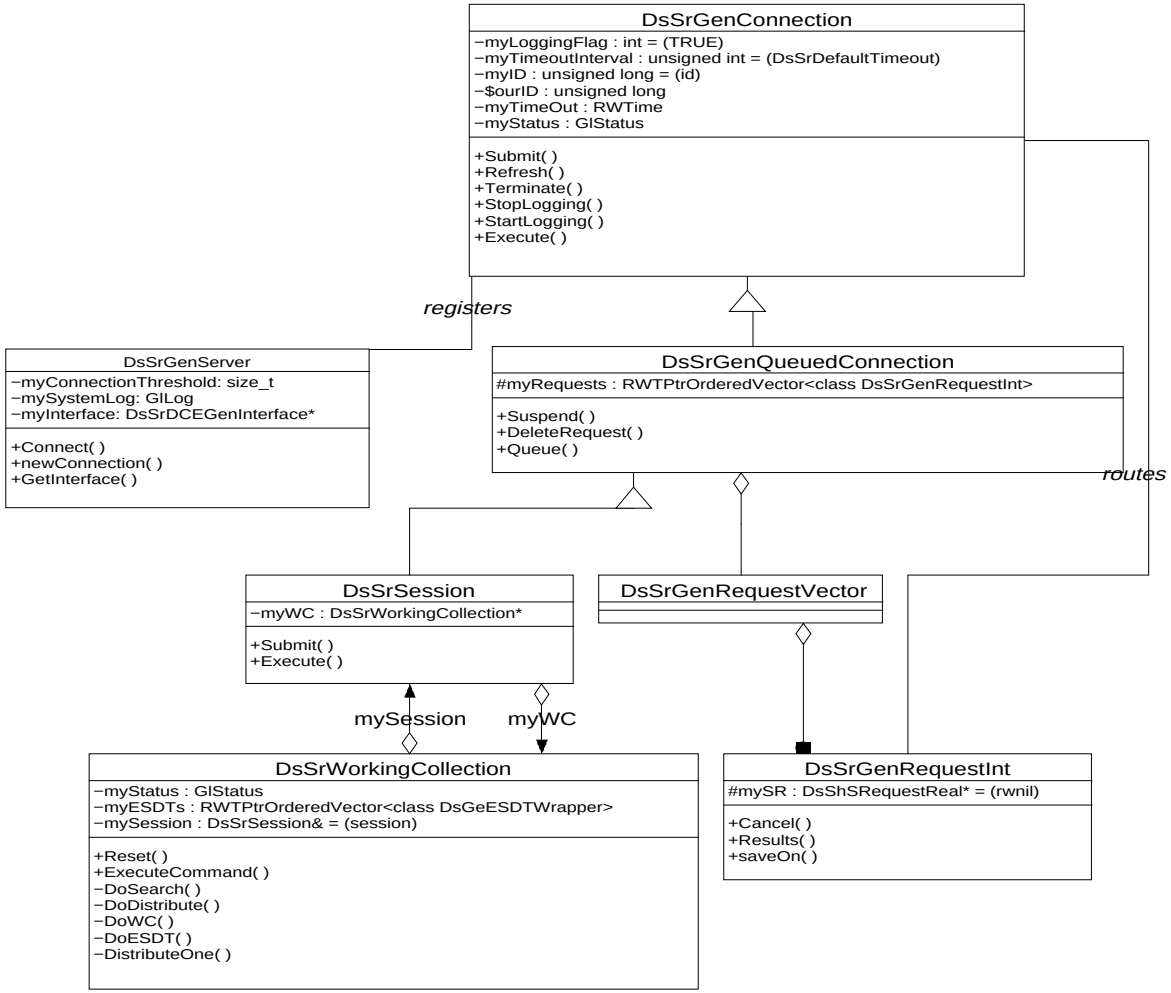


Figure 54 DsSr3

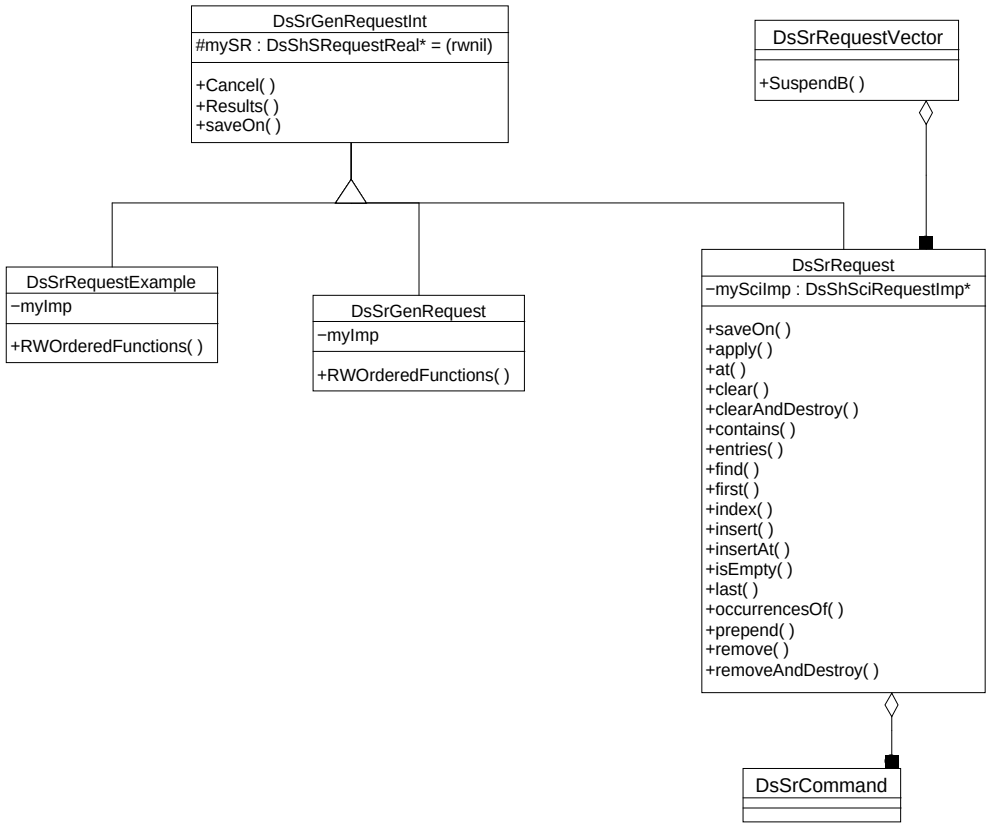


Figure 55 DsSr4

4.3.33.2 sr Classes

4.3.33.2.1 DsSrArchiveCostB Class

Overview:

Distributed Object: No This class represents the archive costs associated with a service. This class represents the cost to put the requested data into or remove the requested data from the storage archive.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrResourceB

Attributes:

Constructors and Destructor:

Operations:

4.3.33.2.2 DsSrClient Class

Overview:

Distributed Object: No Acts as an interface class to the CSMS subsystem to use the accounting and user profiling capabilities.

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myProfile: MSS_UserProifle*`

A pointer to the profile for this client, obtained from MSS.

`mySystemLog: GlLog`

Identifies the log that is used by DsSrClient to log all activity by the object.

Constructors and Destructor:

```
DsSrClient(MSS_UserProfile*);
```

Constructs a client from the given profile.

```
~DsSrClient();
```

Destroys this client.

Operations:

- ConvertToDollarsB

```
RWBoolean ConvertToDollarsB(DsSrRequest & rq);
```

Used to convert the resource usage of a request to its cost.

- DebitB

```
void DebitB(float amt);
```

Used to subtract the given cost from the user's balance.

- GetBalanceB

```
float GetBalanceB();
```

Retrieves the current account balance for this user.

- GetProfile

```
MSS_UserProfile GetProfile();
```

Retrieves the contents of the profile for the user.

- SufficientFundsB

```
RWBoolean SufficientFundsB(DsSrRequest & rq);
```

Used to determine if the user has enough money to perform the request.

4.3.33.2.3 DsSrCommand Class

Overview:

Declarations for the DsSrCommand class.

Distributed Object: No This class supplies the interface needed by other server objects to the core command information. An implementation of the DsCommand abstract base class that provides services necessary for data server creation and storage of command objects.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShSciCommandImp

Attributes:

Constructors and Destructor:

```
DsSrCommand(DsSrCommandInfo*);
```

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.

```
~DsSrCommand();
```

Used to destroy a command.

Operations:

4.3.33.2.4 DsSrConnectionMaker Class

Overview:

Declartions for the DsSrConnectionMaker class. This class provides a specialized NewConnection function for DsShDCEConnectionConcrete to create DsSrSession objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShDCEConnectionConcrete

Attributes:

Constructors and Destructor:

```
DsSrConnectionMaker();
```

default constructor

```
DsSrConnectionMaker(const DsSrConnectionMaker& );
```

prevent copying or assigning

```
virtual  ~DsSrConnectionMaker();
```

destructor BEGIN_PROLOG

NAME ~DsSrConnectionMaker

DESCRIPTION

RETURNS nothing

END_PROLOG

Operations:

- NewConnection

```
virtual DsShConnectionReal* NewConnection();
```

specialized to create DsSrSession objects BEGIN_PROLOG

NAME NewConnection

DESCRIPTION create a new DsSrSession

RETURNS a new DsSrSession

END_PROLOG

- operator =

```
DsSrConnectionMaker& operator =(const DsSrConnectionMaker& );
```

4.3.33.2.5 DsSrCostB Class

Overview:

Distributed Object: No This class represents the collection of resource usage information which is provided to ascertain the cost of executing a given service.

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
numResources: int
```

The number of resources included in this cost.

Constructors and Destructor:

```
~DsSrCost();
```

Destructor for DsSrCost objects.

Operations:

- DsSrCost

```
DsSrCost();
```

Constructor for DsSrCost objects.

- FindResource

```
* DsSrResource FindResource(* String);
```

This returns a pointer to a resource which matches the supplied resource identified as a string.

- FirstResource

```
* DsSrResource FirstResource();
```

This returns a pointer to the first resource in the cost.

- NextResource

```
* DsSrResource NextResource();
```

This returns a pointer to the next resource in the cost.

- NumResources

```
int NumResources();
```

This returns the number of resources in the cost.

- operator+

```
* DsSrResource operator+(* DsSrResource);
```

This is the overloading of the "+" operator. This is provided so Cost objects may be added together.

4.3.33.2.6 DsSrCostPolicyB Class

Overview:

Distributed Object: No This class provides the interface to the SDSRV costing mechanism. This abstract class exists in order to support extensions to the types of costing policies that might be used in the Data Server.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
void ~DsSrCostPolicy();
```

CostPolicy destructor.

Operations:

- DsSrCostPolicy

```
void DsSrCostPolicy();
```

The DsSrCostPolicy constructor.

- GetCost

```
const DsSrCost GetCost();
```

This virtual member function returns a Cost object based on the provided service.

4.3.33.2.7 DsSrCostTableB Class

Overview:

Distributed Object: No This class maintains the information about the costs of services in terms of resource usage. This type of CostPolicy provides a mechanism to provide static costs based on services.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrCostPolicyB

Attributes:

Constructors and Destructor:

```
void ~DsSrCostTable();
```

Destructor for DsSrCostTable.

Operations:

- DsSrCostTable

```
void DsSrCostTable();
```

Constructs a CostTable object.

- GetCost

```
DsSrCost GetCost(const * String);
```

This member function returns a Cost object based upon the identified service (provided as a string parameter).

4.3.33.2.8 DsSrCPUUtilizationB Class

Overview:

Distributed Object: No This class represents the CPU utilization associated with a service. This class represents the cost in CPU cycles to perform the requested service.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrResourceB

Attributes:

Constructors and Destructor:

Operations:

4.3.33.2.9 DsSrDescriptorMaker Class

Overview:

Declartions for the DsSrDescriptorMaker class. This class provides a specialized NewDescriptor function for DsShDCEDescriptorConcrete to create DsDeDescriptor objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShDCEDescriptorConcrete

Attributes:

Constructors and Destructor:

```
DsSrDescriptorMaker();
```

default constructor

```
DsSrDescriptorMaker(const DsSrDescriptorMaker& );
```

prevent copying or assigning

```
virtual  ~DsSrDescriptorMaker();
```

destructor BEGIN_PROLOG

NAME ~DsSrDescriptorMaker

DESCRIPTION

RETURNS nothing

END_PROLOG

Operations:

- NewDescriptor

```
virtual DsShDescriptorReal* NewDescriptor(const RWCString& name,
const RWCString& vers);
```

specialized to create DsDeDescriptor objects BEGIN_PROLOG

NAME NewDescriptor

DESCRIPTION create a new DsDeDescriptor

RETURNS a new DsDeDescriptor

END_PROLOG

- operator =

```
DsSrDescriptorMaker& operator =(const DsSrDescriptorMaker& );
```

4.3.33.2.10 DsSrDiskUtilizationB Class

Overview:

Distributed Object: No This class represents the disk storage costs associated with a service. This class represents the amount of disk space is required for the requested data.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrResourceB

Attributes:

Constructors and Destructor:

Operations:

4.3.33.2.11 DsSrEnvVariable Class

Overview:

Distributed Object: No

BEGIN_PROLOG

NAME DsSrEnv.h

DESCRIPTION This class provides the ability to check and get enviroment variables that are required buy DSS operations.

END_PROLOG

Export Control: Implementation

Inheritance Relationships:

Attributes:

`name: RWCString`

`value: RWCString`

Constructors and Destructor:

Operations:

4.3.33.2.12 DsSrEnv Class

Overview:

This class provides the ability to check and get enviroment variables that are required buy DSS operations.

DsSrEnv() Protected constructor. Use Instance() to instantiate a pointer to this class. ~DsSrEnv() Destructor for this class. Deletes the Instance pointer. Instance() If the pointer is uninstantiated it news one and returns it; otherwise it returns the existing one. CheckEnv() Returns a GlStatus. SUCCESS if all set else FAILURE. GetConfigFile() Returns the pathname in the DSSCONFIG file. GetDLLDir() Returns the pathname in DSSDLLDIR. GetStageDir() Returns the pathname in DSSStageDir. GetRetrieveDir() Returns the pathname in DSSSRETRIEVE GetArchiveDir() Returns the pathname in DSSSTARCHIVE GetSybaseDir() Rerurns the pathname in SYBASE GetDBUsername() Returns the name of the DBUSERNAME GetDBPasswd() Returns the password in DBPASSWD GetDsQuery() Returns the name of the database server in DSQUERY GetSybaseInterfaces() Returns the pathname in SYBINTERFACES GetMissingVariables Returns the names of the missing variables SetValue(DsSrEnvVariable*&, const RWCString) Helper function that does the actual assignment of an Environment variable.

DIstributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myInstance: DsSrEnv*`

`myMissingVariables: RWCString`

myVariables: RWTValOrderedVector<class DsSrEnvVariable*>

missing environments

Constructors and Destructor:

DsSrEnv();

BEGIN_PROLOG

NAME DsSrEnv()

DESCRIPTION Constructor for this class. Assigns Environment variables to attributes. If the Environment variable does not exist, a null string.

RETURNS None

END_PROLOG

~DsSrEnv();

constructor BEGIN_PROLOG

NAME ~DsSrEnv()

DESCRIPTION Destructor for this class. Deletes the pointer to the instance.

RETURNS None

END_PROLOG

Operations:

- CheckDSSEnv

GlStatus CheckDSSEnv();

returns SYBINTERFACE BEGIN_PROLOG

NAME CheckDSSEnv()

DESCRIPTION Checks that all the Environment variables for DSS are set. If any any ar not set FAILURE is returned.

RETURNS GlStatus SUCCESS if all are set. FAILURE if any are not set.

NOTES The caller of this routine is responsible for deleting the GlStatus* that is allocated.

END_PROLOG

- GetArchiveDir

RWCString GetArchiveDir();

returns DSSSTRETRIEVE BEGIN_PROLOG

NAME GetArchiveDir()

DESCRIPTION Returns the directory used for archiving files. If the environment variable is not set, it is added to myMissingVariables

RETURNS RWCString The directory used for archiving files.

END_PROLOG

- GetConfigFile

  `RWCString GetConfigFile();`

  return pointer to the instance BEGIN_PROLOG

  NAME GetConfigFile()

  DESCRIPTION Returns the location DataServer configuration file. If the environment variable is not set, it is added to myMissingVariables

  RETURNS RWCString The location of the configuration file.

  END_PROLOG

- GetDBPasswd

  `RWCString GetDBPasswd();`

  returns DBUSERNAME BEGIN_PROLOG

  NAME GetDBPasswd()

  DESCRIPTION Returns the Password of the Database user. If the environment variable is not set, it is added to myMissingVariables

  RETURNS RWCString The password of the database user.

  END_PROLOG

- GetDBUsername

  `RWCString GetDBUsername();`

  retunrs SYBASE BEGIN_PROLOG

  NAME GetDBUsername()

  DESCRIPTION Returns the name of the database user. If the environment variable is not set, it is added to myMissingVariables

  RETURNS RWCString The name of the database user.

  END_PROLOG

- GetDLLDir

  `RWCString GetDLLDir();`

  returns DSSCONFIG BEGIN_PROLOG

  NAME GetDLLDir()

  DESCRIPTION Returns the location of the Dynamic Library Directory. If the environment variable is not set, it is added to myMissingVariables

  RETURNS RWCString The location of the DLL Directory.

  END_PROLOG

- GetDsQuery

  `RWCString GetDsQuery();`

  returns DBPASSWD BEGIN_PROLOG

  NAME GetDsQuery()

  DESCRIPTION Returns the name of the database server. If the environment variable is not set, it is added to myMissingVariables

  RETURNS RWCString The name of the database server.

  END_PROLOG

- GetMissingVariables

  `const RWCString GetMissingVariables() const;`

  returns TRUE if all envs set else returns FALSE BEGIN_PROLOG

  NAME GetMissingVariables()

  DESCRIPTION Returns the names of the environment variables that are not set.

  RETURNS RWCString The names of the envirnment variables that are not set.

  END_PROLOG

- GetRetrieveDir

  `RWCString GetRetrieveDir();`

  returns DSSSSTAGEDIR BEGIN_PROLOG

  NAME GetRetrieveDir()

  DESCRIPTION Returns the directory for files being retrieved from the archive. If the environment variable is not set, it is added to myMissingVariables

  RETURNS RWCString The directory for files being retrieved from the archive.

  END_PROLOG

- GetStageDir

  `RWCString GetStageDir();`

  returns DLLDIR BEGIN_PROLOG

  NAME GetStageDir()

  DESCRIPTION Returns the location of the directory used to stage data files. If the environment variable is not set, it is added to myMissingVariables

  RETURNS RWCString The location of the directory used to stage data files.

  END_PROLOG

- GetSybaseDir

  `RWCString GetSybaseDir();`

returns DSSSTARCHIVE BEGIN_PROLOG

NAME GetSybaseDir()

DESCRIPTION Returns the Sybase location. If the environment variable is not set, it is added to myMissingVariables

RETURNS RWCString The location of Sybase.

END_PROLOG

- GetSybaseInterfaces

```
RWCString GetSybaseInterfaces();
```

returns DSQUERY BEGIN_PROLOG

NAME GetSybaseInterfaces()

DESCRIPTION Returns the name of the sybase interfaces file. If the environment variable is not set, it is added to myMissingVariables

RETURNS RWCString The name of the sybase interfaces file.

END_PROLOG

- Instance

```
static DsSrEnv* Instance();
```

BEGIN_PROLOG

NAME Instance

DESCRIPTION Return a pointer to the instance of this class if it already exists. If not, it news one and returns it.

RETURNS DsSrEnv* The instance of this class

END_PROLOG

- SetValue

```
void SetValue(DsSrEnvVariable*& staticptr, const RWCString
envar);
```

destructor BEGIN_PROLOG

NAME SetValue(DsSrEnvVariable*&, const RWCString)

DESCRIPTION This is a helper function that is called by the Get functions. It is responsible for allocating space for a new Environment variable and the assign the new value.

RETURNS Nothing

END_PROLOG

4.3.33.2.13 DsSrFixedPersonnelCostB Class

Overview:

Distributed Object: No This class represents fixed personnel costs associated with a service. This class represents the portion of fixed personnel costs that can be assigned to a unit of work.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrResourceB

Attributes:

Constructors and Destructor:

Operations:

4.3.33.2.14 DsSrGenCatalogPool Class

Overview:

Declaration of the DsSrGenCatalogPool class. These are used by data server objects to obtain access to a catalog. A pool of catalogs is maintained by DsSrGenCatalogPool, an example of the singleton pattern. Data server objects should use DsSrGenCatalogWrapper objects to obtain a catalog, rather than directly using DsSrGenCatalogPool.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myCatalogs: RWTValOrderedVector<class DsMdCatalog*>`

mutex for each catalog

`myLock: DCEPthreadMutex`

actual number of catalogs

myMutexs: RWTValOrderedVector<class DCEPthreadMutex*>

access lock for vectors

mySize: int

ourInstance: DsSrGenCatalogPool*

catalogs

Constructors and Destructor:

DsSrGenCatalogPool(int size);

the hidden constructor that allows another pool size BEGIN_PROLOG

NAME DsSrGenCatalogPool

DESCRIPTION Constructor that takes an argument indicating the desired size of the pool of catalogs.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

DsSrGenCatalogPool(const DsSrGenCatalogPool&);

copy constructor (NOT IMPLEMENTED)

~DsSrGenCatalogPool();

destructor BEGIN_PROLOG

NAME ~DsSrGenCatalogPool

DESCRIPTION destructor. Deletes all of the dynamically allocated objects.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- AcquireCatalog

DsMdCatalog* AcquireCatalog();

Used to acquire (and lock) a catalog from the pool BEGIN_PROLOG

NAME AcquireCatalog

DESCRIPTION Obtain and lock a catalog from the pool. If all catalogs are in use, this routine will sleep until one is available.

RETURNS Pointer to catalog

GLOBALS none

FILES none

NOTES While the data server handles requests synchronously, this function is more complex than necessary; in fact, there can only be one pool allocated at a time, so the looping & locking is mostly unexercised. When the server become asynchronous, this function will need to be enhanced to use (e.g.) a condition variable to wait for an available catalog.

END_PROLOG

- Instance

```
static DsSrGenCatalogPool* Instance();
```

static function that returns the single instance BEGIN_PROLOG

NAME Instance

DESCRIPTION Returns the instance of this singleton.

RETURNS Pointer to the instance, or rwnil if one could not be created.

GLOBALS none

FILES none

END_PROLOG

- operator =

```
DsSrGenCatalogPool& operator =(const DsSrGenCatalogPool& );
```

assignment operator (NOT IMPLEMENTED)

- ReleaseCatalog

```
int ReleaseCatalog(DsMdCatalog* cat);
```

used to release (and unlock) a catalog previously acquired BEGIN_PROLOG

NAME ReleaseCatalog

DESCRIPTION Releases a catalog that was previously acquired.

RETURNS TRUE if successful, FALSE otherwise

GLOBALS none

FILES none

END_PROLOG

4.3.33.2.15 DsSrGenCatalogWrapper Class

Overview:

Declaration of the DsSrGenCatalogWrapper class. These are used by data server objects to obtain access to a catalog. Data server objects can obtain a catalog by simply instantiating a DsSrGenCatalogWrapper object; this method is ensures that an allocated catalog will always be released, even in the presence of exceptions.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myCatalog: DsMdCatalog*
```

Constructors and Destructor:

```
DsSrGenCatalogWrapper();
```

default constructor

```
DsSrGenCatalogWrapper(const DsSrGenCatalogWrapper& );
```

copy constructor (NOT IMPLEMENTED)

```
~DsSrGenCatalogWrapper();
```

destructor - automatically unlocks BEGIN_PROLOG

NAME ~DsSrGenCatalogWrapper

DESCRIPTION destructor. Releases catalog if one has been acquired.

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- Get

```
DsMdCatalog* Get();
```

obtains (locked) catalog BEGIN_PROLOG

NAME Get

DESCRIPTION Obtains a (locked) catalog from the DsSrGenCatalogPool singleton, or returns one that has already been obtained.

RETURNS Pointer to the catalog

GLOBALS none

FILES none

NOTES Unless an error occurs, a catalog will always be obtained. If all catalogs are in use, this call will wait until one becomes available.

END_PROLOG

- operator =

```
DsSrGenCatalogWrapper& operator =(const DsSrGenCatalogWrapper&
);
```

assignment operator (NOT IMPLEMENTED)

- Release

```
void Release();
```

releases (unlocks) previously locked catalog BEGIN_PROLOG

NAME Release

DESCRIPTION Releases a catalog (if one has been acquired).

RETURNS

GLOBALS none

FILES none

END_PROLOG

4.3.33.2.16 DsSrGenConnection Class

Overview:

Declaration of the DsSrGenConnection class. This is the (distributed) base class for all connections/sessions that exist in the dataserver.

Distributed Object: No in seconds

Export Control: Implementation

Inheritance Relationships:

Inherits from DsShConnectionReal

Attributes:

`myID: unsigned long`

seconds between timeouts

`myLoggingFlag: int`

current status

`myStatus: GlStatus`

time of next timeout

`myTimeoutInterval: unsigned int`

TRUE == logging is on

`myTimeOut: RWTime`

`ourID: unsigned long`

my ID

Constructors and Destructor:

`DsSrGenConnection(unsigned long id);`

default constructor

`DsSrGenConnection(const DsSrGenConnection& );`

copy constructor NOT IMPLEMENTED

`virtual  ~DsSrGenConnection();`

existing connection ID destructor BEGIN_PROLOG

NAME ~DsSrGenConnection

DESCRIPTION destructor

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- Execute

`virtual void Execute(DsSrGenRequestInt& ) = 0;`

execute a request

- GetConnectionID

  ```
  unsigned long GetConnectionID() const;
  ```

  the the ID of this connection BEGIN_PROLOG

  NAME GetConnectionID

  DESCRIPTION Used to obtain the ID of a connection.

  RETURNS the connection ID

  GLOBALS none

  FILES none

  END_PROLOG

- GetStatus

  ```
  GlStatus GetStatus() const;
  ```

  get the current status

- GetTimeOut

  ```
  RWTime GetTimeOut() const;
  ```

  obtain the current timeout

- operator =

  ```
  DsSrGenConnection& operator =(const DsSrGenConnection& );
  ```

  assignment operator NOT IMPLEMENTED

- Refresh

  ```
  virtual void Refresh();
  ```

  local functions: refresh object from configuration BEGIN_PROLOG

  NAME Refresh

  DESCRIPTION Loads (possibly updated) configuration information

  RETURNS nothing

  GLOBALS none

  FILES none

  NOTES Configuration information for connections has yet to be determined, so this function is a noop.

  END_PROLOG

- SetTimeOut

  ```
  void SetTimeOut(unsigned int seconds);
  ```

  set the timeout

- StartLogging

  ```
  void StartLogging();
  ```

  start logging

- StopLogging

  ```
  void StopLogging();
  ```

  stop logging

- Submit

  ```
  virtual void Submit(DsShSRequestReal* ) = 0;
  ```

  distributed functions overridden:

- Terminate

  ```
  virtual GlStatus Terminate();
  ```

  terminate a connection BEGIN_PROLOG

  NAME Terminate

  DESCRIPTION An external, asynchronous method of terminating a connection.

  RETURNS status

  GLOBALS none

  FILES none

  NOTES Will be implemented in conjunction with operator interface.

  END_PROLOG

4.3.33.2.17 DsSrGenQueuedConnection Class

Overview:

Declaration of the class DsSrGenQueuedConnection. This specialization of the DsSrGenConnection class adds the ability to queue requests as they arrive.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrGenConnection

Attributes:

```
myRequests: RWTPtrOrderedVector<class DsSrGenRequestInt>
```

Constructors and Destructor:

```
DsSrGenQueuedConnection(unsigned long cid);
```

default constructor

```
DsSrGenQueuedConnection(const DsSrGenQueuedConnection& );
```

copy constructor (NOT IMPLEMENTED)

```
~DsSrGenQueuedConnection();
```

existing connection or 0 destructor BEGIN_PROLOG

NAME ~DsSrGenQueuedConnection

DESCRIPTION destructor

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- DeleteRequest

```
GlStatus DeleteRequest(DsSrGenRequestInt* );
```

the connection ID delete a queued request

- operator =

```
DsSrGenQueuedConnection& operator =(const
DsSrGenQueuedConnection& );
```

assignment operator (NOT IMPLEMENTED)

- Queue

```
virtual GlStatus Queue(DsSrGenRequestInt* newreq);
```

the request to delete queue a request

- Suspend

```
GlStatus Suspend(unsigned long& );
```

suspend this connection

4.3.33.2.18 DsSrGenRequestInt Class

Overview:

Declaration of the class DsSrGenRequestInt. The server-side object that represents a request to the data server. Consists of a sequence of DsSrCommand objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
mySR: DsShSRequestReal*
```

Constructors and Destructor:

```
DsSrGenRequestInt(const DsSrGenRequestInt& );
```

copy constructor NOT IMPLEMENTED

```
DsSrGenRequestInt();
```

default contructor

```
DsSrGenRequestInt(DsShSRequestReal* sr);
```

normal constructor

```
virtual  ~DsSrGenRequestInt();
```

associated distributed request destructor BEGIN_PROLOG
NAME ~DsSrGenRequestInt
DESCRIPTION destructor. Destroys the contents of self (vector).
RETURNS nothing
END_PROLOG

Operations:

- Cancel

```
virtual int Cancel();
```

request to compare cancel a request BEGIN_PROLOG

NAME Cancel

DESCRIPTION Used to cancel the request.

RETURNS boolean; false means request was already active and could not be canceled.

END_PROLOG

- GetRequestImp

```
virtual DsShGenRequestImp* GetRequestImp() = 0;
```

get a pointer to the request implementation object

- GetRequestImp

```
virtual const DsShGenRequestImp* GetRequestImp() const = 0;
```

get a pointer to the request implementation object (const version)

- GetServerType

```
const RWCString& GetServerType() const;
```

any output stream get the server type BEGIN_PROLOG

NAME GetServerType

DESCRIPTION returns the server type

RETURNS server type

END_PROLOG

- GetStatus

```
const GlStatus& GetStatus() const;
```

get the current status BEGIN_PROLOG

NAME GetStatus

DESCRIPTION Used to obtain the current status.

RETURNS Constant reference to the status object.

END_PROLOG

- operator ==

```
int operator ==(const DsSrGenRequestInt& );
```

comparison operator BEGIN_PROLOG

NAME operator==

DESCRIPTION Comparison operator. Necessary for this class to be instantiated in an RW vector template. Doesn't really provide useful comparison functionality.

RETURNS boolean

END_PROLOG

- operator =

  `DsSrGenRequestInt& operator =(const DsSrGenRequestInt& );`

  assignment operator NOT IMPLEMENTED

- Results

  `virtual GlParameterList& Results();`

  status to set get a reference to the results BEGIN_PROLOG

  NAME Results

  DESCRIPTION Used to obtain a (mutable) reference to the results object for this request.

  RETURNS The reference.

  END_PROLOG

- saveOn

  `virtual void saveOn(ostream& os) const;`

  dumps human-readable text of an object BEGIN_PROLOG

  NAME saveOn

  DESCRIPTION Write to a normal, human-readable stream.

  RETURNS nothing

  END_PROLOG

- SetStatus

  `virtual void SetStatus(const GlStatus& stat);`

  set the status BEGIN_PROLOG

  NAME SetStatus

  DESCRIPTION Used to set the status.

  RETURNS nothing

  END_PROLOG

4.3.33.2.19 DsSrGenRequestVector Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.3.33.2.20 DsSrGenRequest Class

Overview:

Declaration of the class DsSrGenRequestInt. The server-side object that represents a request to the data server. Consists of a sequence of DsSrCommand objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrGenRequestInt

Attributes:

myImp:

Constructors and Destructor:

Operations:

- GetRequestImp

```
GetRequestImp();
```

- RWOrderedFunctions

```
RWOrderedFunctions();
```

4.3.33.2.21 DsSrGenServer Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myConnectionThreshold: size_t:

myInterface: DsSrDCEGenInterface*:

mySystemLog: GlLog:
```

Constructors and Destructor:

```
DsSrGenServer();

~DsSrGenServer();
```

Operations:

- Connect

```
Connect();
```

- DsSrGenServerExample

```
DsSrGenServerExample();
```

- GetConnectionThreshold

```
GetConnectionThreshold();
```

- GetInterface

```
GetInterface();
```

- newConnection

```
newConnection();
```

- SetConnectionThreshold

```
SetConnectionThreshold();
```

4.3.33.2.22 DsSrIOUtililizationB Class

Overview:

Distributed Object: No This class represents the IO costs associated with a service. This class represents the amount of IO usage which is required to perform the requested service.

Export Control: Public

Inheritance Relationships:

Inherits from DsSrResourceB

Attributes:

Constructors and Destructor:

Operations:

4.3.33.2.23 DsSrMediaUtilizationB Class

Overview:

Distributed Object: No This class represents the Media costs associated with a service. This class represents the amount of media utilization other than disk which is required for the requested service.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrResourceB

Attributes:

Constructors and Destructor:

Operations:

4.3.33.2.24 DsSrRequestExample Class

Overview:

Declaration of the class DsSrRequestExample. The server-side object that represents a request to the data server. Consists of a sequence of DsSrCommand objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrGenRequestInt

Attributes:

`myImp`:

Constructors and Destructor:

Operations:

- DsSrGenRequest

```
DsSrGenRequest();
```

- GetRequestImp

```
GetRequestImp();
```

- RWOrderedFunctions

```
RWOrderedFunctions();
```

4.3.33.2.25 DsSrRequestVector Class

Overview:

Distributed Object: No Manages multiple DsSrRequests for a DsSrQueuedConnection objects. Provides ordering based on priority of DsSrRequests and order of receipt (FIFO).

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

```
DsSrRequestVector();
```

Constructs an empty request vector.

```
~DsSrRequestVector();
```

Destroys this request vector.

Operations:

- DsSrRequestVectorB

```
DsSrRequestVectorB(GlUR& sessionID);
```

This version of the constructor restores a previously suspended session (identified by the GlUR sessionID parameter).

- SuspendB

```
GlStatus& SuspendB();
```

This operation iterates over the requests in the queue, disconnecting each in turn.

4.3.33.2.26 DsSrRequest Class

Overview:

Declaration of the class DsSrRequest. The server-side object that represents a request to the data server.

Declaration of the class DsSrRequestExample. The server-side object that represents a request to the data server. Consists of a sequence of DsSrCommand objects.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrGenRequestInt

Attributes:

```
mySciImp: DsShSciRequestImp*
```

Constructors and Destructor:

```
DsSrRequest(DsShSRequestReal* sr);
```

normal constructor

```
DsSrRequest(const DsSrRequest& );
```

copy constructor NOT IMPLEMENTED

```
~DsSrRequest();
```

associated distributed request destructor BEGIN_PROLOG

NAME ~DsSrRequest

DESCRIPTION destructor. Destroys the contents of self (vector).

RETURNS nothing

END_PROLOG

Operations:

- apply

```
void apply(void (*) (class RWCollectable*, void*) ap, void* x);
```

- at

```
DsShSciCommandImp*& at(unsigned int i);
```

- at

```
const DsShSciCommandImp* at(unsigned int i) const;
```

- clearAndDestroy

```
void clearAndDestroy();
```

- clear

```
void clear();
```

- contains

```
int contains(const DsShSciCommandImp* p) const;
```

- entries

```
unsigned int entries() const;
```

- find

```
DsShSciCommandImp* find(const DsShSciCommandImp* p) const;
```

- first

```
DsShSciCommandImp* first() const;
```

- GetDomain

```
const RWTPtrOrderedVector<class GlUR>& GetDomain() const;
```

get the domain BEGIN_PROLOG

NAME GetDomain

DESCRIPTION Used to obtain the request's domain

RETURNS UR vector of ESDTs included in this requests domain. This vector is empty if the domain is "all ESDTs in the working collection".

END_PROLOG

- GetPriority

  ```
  enum DsShGlobal::DsEShSciPriority GetPriority() const;
  ```

  get the priority BEGIN_PROLOG
  NAME GetPriority
  DESCRIPTION Used to obtain the request's priority.
  RETURNS The priority, as given by the base object.
  END_PROLOG

- GetRequestImp

  ```
  virtual DsShGenRequestImp* GetRequestImp();
  ```

  used to obtain the implementation BEGIN_PROLOG
  NAME GetRequestImp
  DESCRIPTION Used to obtain the request's implementation
  RETURNS pointer to the implementation object
  END_PROLOG

- GetRequestImp

  ```
  virtual const DsShGenRequestImp* GetRequestImp() const;
  ```

  used to obtaint the implementation (const version) BEGIN_PROLOG
  NAME GetRequestImp
  DESCRIPTION Used to obtain the request's implementation
  RETURNS constant pointer to the implementation object
  END_PROLOG

- index

  ```
  unsigned int index(const DsShSciCommandImp* p) const;
  ```

- insertAt

  ```
  void insertAt(unsigned int i, DsShSciCommandImp* p);
  ```

- insert

  ```
  DsShSciCommandImp* insert(DsShSciCommandImp* p);
  ```

- isEmpty

```
int isEmpty() const;
```

- last

```
DsShSciCommandImp* last() const;
```

- occurrencesOf

```
unsigned int occurrencesOf(const DsShSciCommandImp* p) const;
```

- operator =

```
DsSrRequest& operator =(const DsSrRequest& );
```

assignment operator NOT IMPLEMENTED

- operator []

```
DsShSciCommandImp*& operator [](unsigned int i);
```

any output stream macro that defines all of the vector command-access methods

- prepend

```
DsShSciCommandImp* prepend(DsShSciCommandImp* p);
```

- removeAndDestroy

```
void removeAndDestroy(const DsShSciCommandImp* p);
```

- remove

```
DsShSciCommandImp* remove(const DsShSciCommandImp* p);
```

- saveOn

```
virtual void saveOn(ostream& os) const;
```

dumps human-readable text of an object BEGIN_PROLOG
NAME saveOn
DESCRIPTION Write to a normal, human-readable stream.
RETURNS nothing
END_PROLOG

4.3.33.2.27 DsSrResourceB Class

Overview:

Distributed Object: No This abstract class represents the interface for various types of resources that are utilized and included in service costs. This base class provides the common information about and operations on system resources.

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
name: * String

units: * String

utilization: int
```

Constructors and Destructor:

```
~DsSrResource();
```

The destructor for DsSrResource objects.

Operations:

- DsSrResource

```
DsSrResource();
```

The constructor for DsSrResource objects.

- GetName

```
const * String GetName();
```

This service returns the name of the resource.

- GetUnits

```
const * String GetUnits();
```

This method returns the units used by this resource.

- GetUtilization

  ```
  const * int GetUtilization();
  ```

  This method returns the actual amount of resource used.

- operator+

  ```
  * DsSrResource operator+(* DsSrResource);
  ```

  This overloading of the "+" operator is provided to enable specific types of resources to be added together.

4.3.33.2.28 DsSrServer Class

Overview:

Declaration of the server class. This is an example of a singleton pattern, using a static function to ensure that only a single instance of this class exists in a given executable.

Distributed Object: No 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.

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myConnectionThreshold: int
```

The maximum number of connections that can be started by this server.

```
mySystemLog: GlLog
```

Used to log general system activities.

Constructors and Destructor:

```
DsSrServer(DsSrConfig cf);
```

Used to construct the DsSrServer object using the given configuration.

```
~DsSrServer();
```

Used to destroy the server object.

Operations:

- Connect

  ```
  GlStatus Connect(DsSrConnection &);
  ```

  Used to register a connection for this server.

- DeleteConnection

  ```
  GlStatus DeleteConnection(DsSrConnection& conn);
  ```

  Used to terminate and delete an existing connection.

- GetConnectionThreshold

  ```
  int GetConnectionThreshold();
  ```

  Used to retrieve the current connection threshold.

- GetSRVector

  ```
  DsSrSubmittedRequestVector & GetSRVector();
  ```

  Used to obtain a reference to the vector of all active submitted requests. The vector can be modified or searched by the caller.

- ListConnections

  ```
  const DsSrConnectionVector ListConnections();
  ```

  Returns a list of all (currently active) connections known by the server.

- ReportStatus

  ```
  GlStatus ReportStatus();
  ```

  Returns the current status of the server.

- ResumeB

  ```
  GlStatus ResumeB();
  ```

  This operation sets the newly created (from a previously suspended) session to "active" state, so that it will begin to accept new requests.

- SetConnectionThreshold

  ```
  void SetConnectionThreshold(int);
  ```

  Used to set the maximum number of connections that can be active for this server.

- SuspendB

  ```
  GlStatus SuspendB();
  ```

  This operation manages the suspension of the server-side of a user session.

4.3.33.2.29 DsSrSession Class

Overview:

Declaration of the class DsSrSession. A specialization of the DsSrGenQueuedConnection class that adds DsSrWorkingCollection-oriented behavior to the connection.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Inherits from DsSrGenQueuedConnection

Attributes:

```
myWC: DsSrWorkingCollection*
```

Constructors and Destructor:

```
DsSrSession(unsigned long id);
```

default constructor

```
DsSrSession(const DsSrSession& );
```

copy constructor (NOT IMPLEMENTED)

```
virtual  ~DsSrSession();
```

existing connection or 0 destructor BEGIN_PROLOG

NAME ~DsSrSession

DESCRIPTION destructor

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- Execute

```
virtual void Execute(DsSrGenRequestInt& genreq);
```

execute a request BEGIN_PROLOG

NAME Execute

DESCRIPTION Used to execute a request. Again, when the server executes request asynchronously, this function will become much more complex to support simultaneous execution of multiple non-intersecting requests, etc.

RETURNS nothing

END_PROLOG

- operator =

```
DsSrSession& operator =(const DsSrSession& );
```

assignment operator (NOT IMPLEMENTED)

- ResumeB

```
ResumeB();
```

- Submit

```
virtual void Submit(DsShSRequestReal* sr);
```

submit a request to this session BEGIN_PROLOG

NAME Submit

DESCRIPTION Used to submit a request to the session. When the server becomes asynchronous, execution will be its own thread but for now, execute immediately after submit...

RETURNS status

GLOBALS none

FILES none

END_PROLOG

4.3.33.2.30 DsSrWorkingCollection Class

Overview:

Declaration of the class DsSrWorkingCollection. Maintains a vector of ESDT objects that represents the session's current contents. Executes commands by category.

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

myESDTs: RWTPtrOrderedVector<class DsGeESDTWrapper>

current status

mySession: DsSrSession&

vector of ESDTs

myStatus: GlStatus

Constructors and Destructor:

DsSrWorkingCollection(DsSrSession& session);

normal constructor

DsSrWorkingCollection(const DsSrWorkingCollection&);

copy constructor NOT IMPLEMENTED

~DsSrWorkingCollection();

the associated session destructor BEGIN_PROLOG

NAME ~DsSrWorkingCollection

DESCRIPTION destructor. Destroys anything currently in the collection

RETURNS nothing

END_PROLOG

Operations:

- DistributeOne

GlStatus DistributeOne(const DsShSciCommandImp& cmd, unsigned int esdtidx, unsigned int& filecount);

add one ESDT to the distribution list BEGIN_PROLOG

NAME DistributeOne

DESCRIPTION Add one ESDT to the distribution list

RETURNS status

END_PROLOG

- DoDistribute

GlStatus DoDistribute(const DsShSciCommandImp& cmd, const RWTPtrOrderedVector<class GlUR>& domain, GlParameterList& cmdresults);

where to store results handle distribute commands BEGIN_PROLOG

NAME DoDistribute

DESCRIPTION Handle distribute commands.

RETURNS status

END_PROLOG

- DoESDT

```
GlStatus DoESDT(const DsShSciCommandImp& cmd, const
RWTPtrOrderedVector<class GlUR>& domain, GlParameterList&
cmdresults);
```

handle ESDT commands BEGIN_PROLOG

NAME DoESDT

DESCRIPTION Handle ESDT commands.

RETURNS status

END_PROLOG

- DoSearch

```
GlStatus DoSearch(const DsShSciCommandImp& cmd, GlParameterList&
cmdresults);
```

handle search commands BEGIN_PROLOG

NAME DoSearch

DESCRIPTION Handle search commands.

RETURNS status

END_PROLOG

- DoWC

```
GlStatus DoWC(const DsShSciCommandImp& cmd, GlParameterList&
cmdresults);
```

handle WC commands BEGIN_PROLOG

NAME DoWC

DESCRIPTION Handle WC commands.

RETURNS status

END_PROLOG

- ExecuteCommand

```
GlStatus ExecuteCommand(const DsShSciCommandImp& cmd, const
RWTPtrOrderedVector<class GlUR>& domain, GlParameterList&
cmdresults, DsSrRequest& request);
```

execute a command BEGIN_PROLOG

NAME ExecuteCommand

DESCRIPTION Used to execute a command over the current collection. Examines the category of the command to determine how to handle it. All CATALOG commands are assumed to be searches. All DISTRIBUTION commands are assumed to be acquires. Category WC commands are parsed and executed. ESDT category commands are handed to the ESDTs specified by the given domain. If domain is empty, they are handed to all ESDTs in the collection.

RETURNS status

END_PROLOG

- GetStatus

```
const GlStatus& GetStatus();
```

request for status setting get the status of the working collection

- operator =

```
DsSrWorkingCollection& operator =(const DsSrWorkingCollection&
);
```

assignment operator NOT IMPLEMENTED

- Reset

```
void Reset();
```

reset the collection (i.e. make it empty) BEGIN_PROLOG

NAME Reset

DESCRIPTION clears the collection

RETURNS nothing

END_PROLOG

4.3.34 sw Class Category

4.3.34.1 Overview

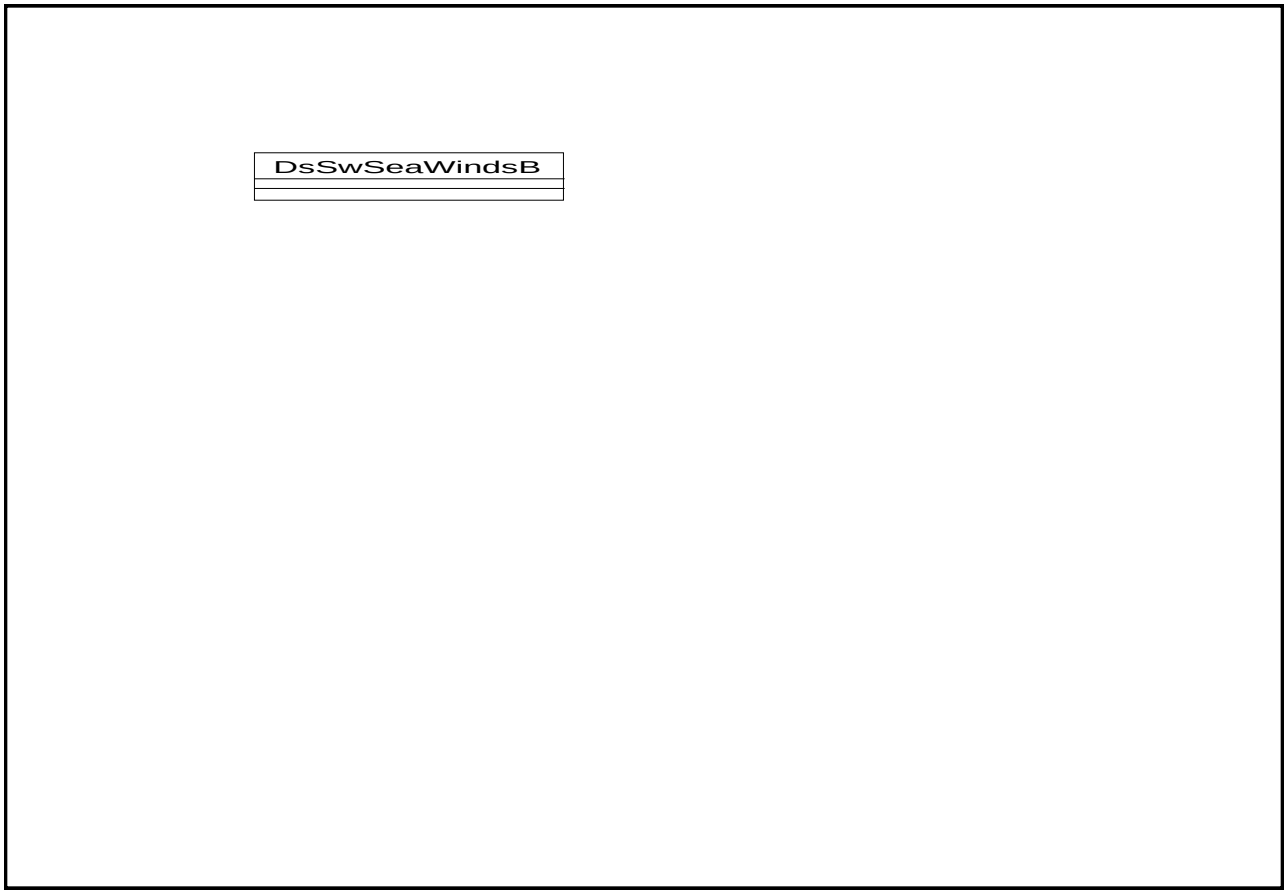


Figure 50 Main

4.3.34.2 sw Classes

4.3.34.2.1 DsSwSeaWindsB Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the SeaWinds Instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsGeECSDataProduct

Attributes:

Constructors and Destructor:

```
DsSwSeaWindsB();

~DsSwSeaWindsB();
```

Operations:

4.3.35 tm Class Category

4.3.35.1 Overview

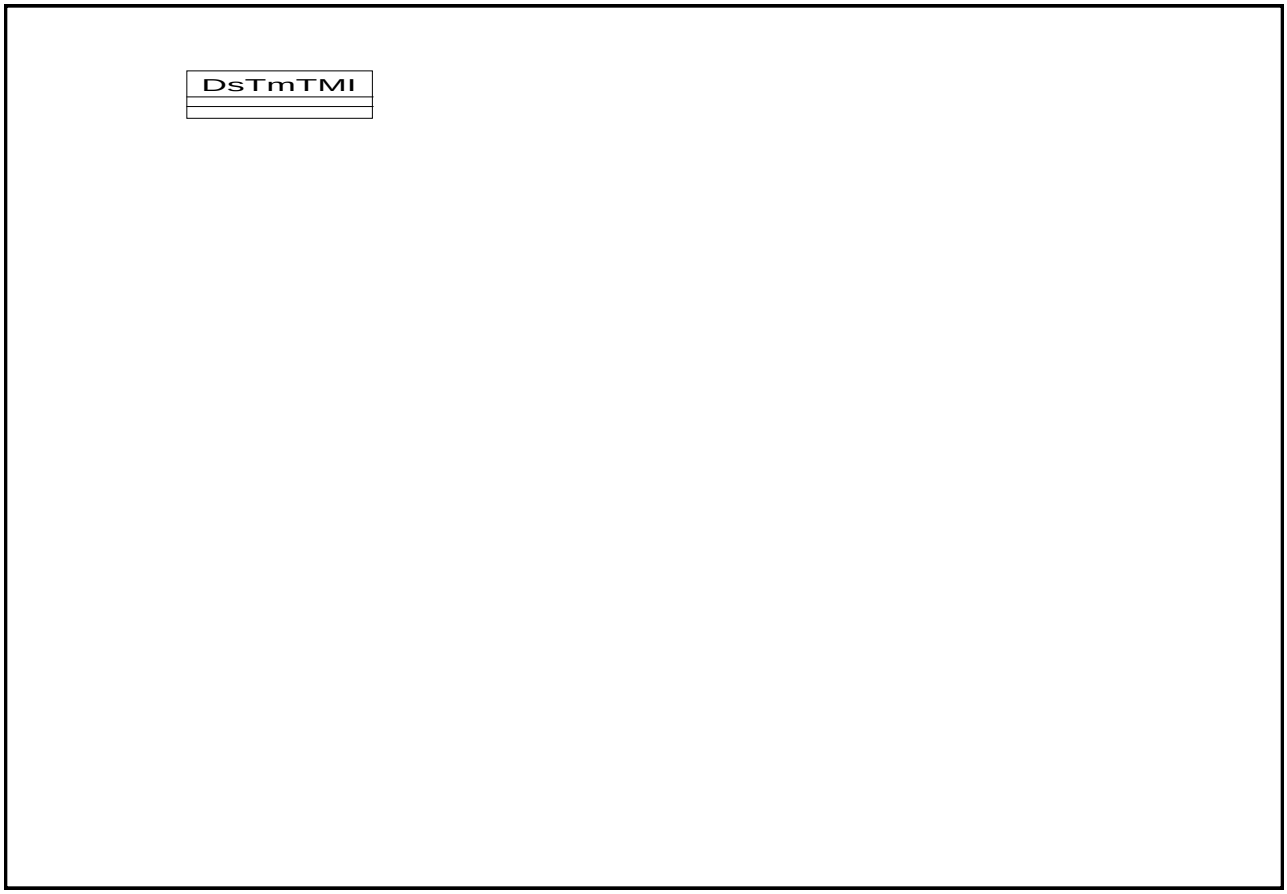


Figure 57 Main

4.3.35.2 tm Classes

4.3.35.2.1 DsTmTMI Class

Overview:

Distributed Object: No This class is an abstraction of the products derived directly from the TRMM platform Microwave Imager (TMI) instrument data.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

```
DsTmTMI();

~DsTmTMI();
```

Operations:

4.3.36 vi Class Category

4.3.36.1 Overview

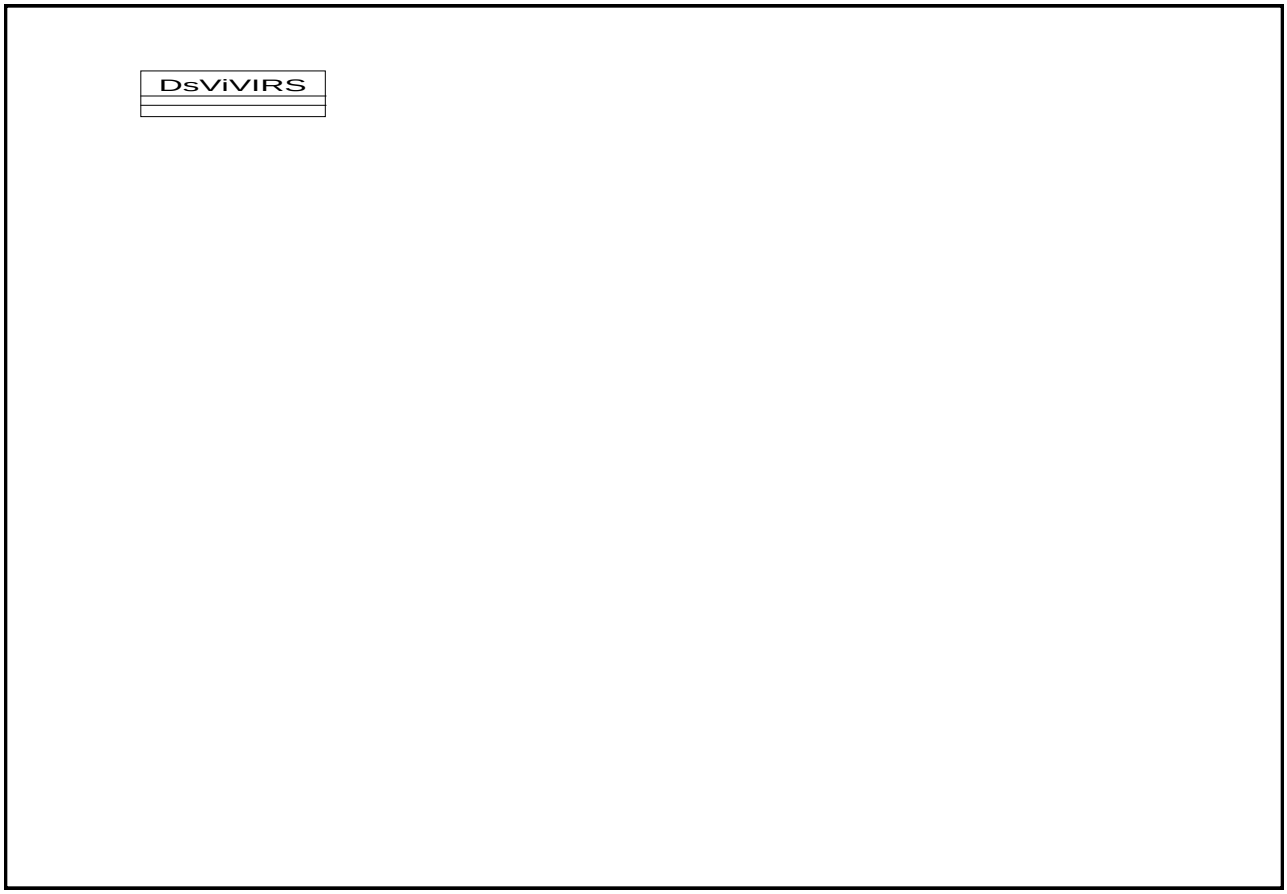


Figure 58 Main

4.3.36.2 vi Classes

4.3.36.2.1 DsViVIRS Class

Overview:

Distributed Object: No This class is an abstraction of the products derived from the TRMM platform Visible and Infrared Scanner (VIRS) data.

Export Control: Public

Inheritance Relationships:

Inherits from DsNpNonECSDataProduct

Attributes:

Constructors and Destructor:

```
DsViVIRS();

~DsViVIRS();
```

Operations:

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
- 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
- Client-side Resuming a Session
- Deleting a Queued Request
- Ending Session No Active Requests
- Fulfilling a One-time Subscription
- Fulfilling Open Ended Subscription
- Inserting a New ESDT
- Inserting Composite 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
- Server-side Resuming a Session
- Startup of a Science Data Server

- Submitting a Subscription
- Subsetting a granule
- Suspending a Session
- Unregistering a Subscribable Event
- Update Server Configuration
- Updating a Subscription
- Validating Metadata
- Changing Priority for Requests-in-Progress
- Handling Multi-State Granules
- Creating an On-Demand Product
- Client and Server Suspend/Resume a Session <*NEW*>

Figures 4.4-1 through 4.4-42 are event trace diagrams from Rational Rose that define the class interactions used to support these scenarios.

4.5 Return Request Status

4.5.0.1 Summary

This scenario shows how a server side object (service object) updates the status of a submitted request, DsShRequestReal. This general mechanism is used to return request results as well as state changes. The service object represents a generic object providing a service to the request. To update status or provide results, the service object creates an instace of a string parameter, GlStringP and sets the name and value attributes (e.g., “STATUS = DONE”). This must be done for each parameter in the reult set. The set of parameters is held in a GlParameterList, instantiated by the service object. Each parameter is inserted into the parameter list. The request’s SetStatus method is then invoked with the parameter list as an argument. The results are passed to the client software via the SDSRV Asynchronous Status Update scenario.

4.5.0.2 Assumptions

- A request has been submitted and is active.

•The request has created a “service object” to perform some work.

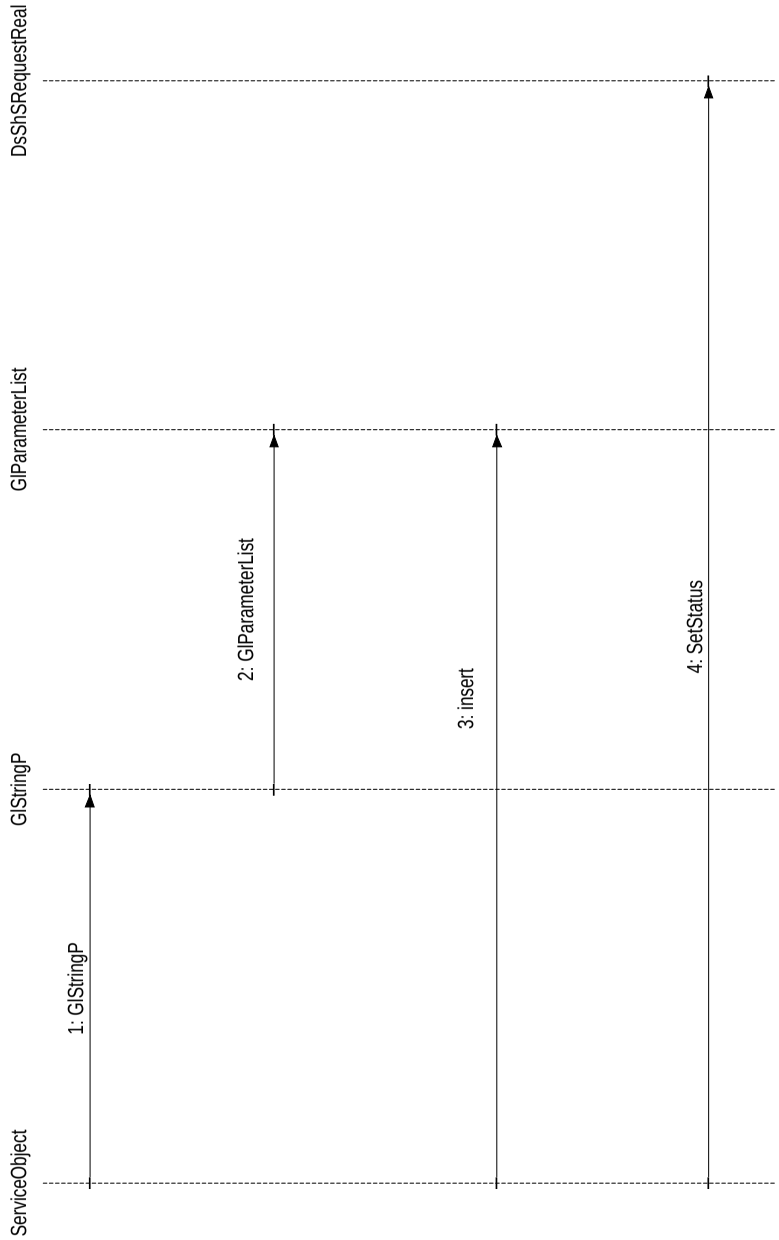


Figure 59 Return Request Status

4.5.1 SDSRV Acquiring An ESDT

4.5.1.1 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.

4.5.1.2 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.

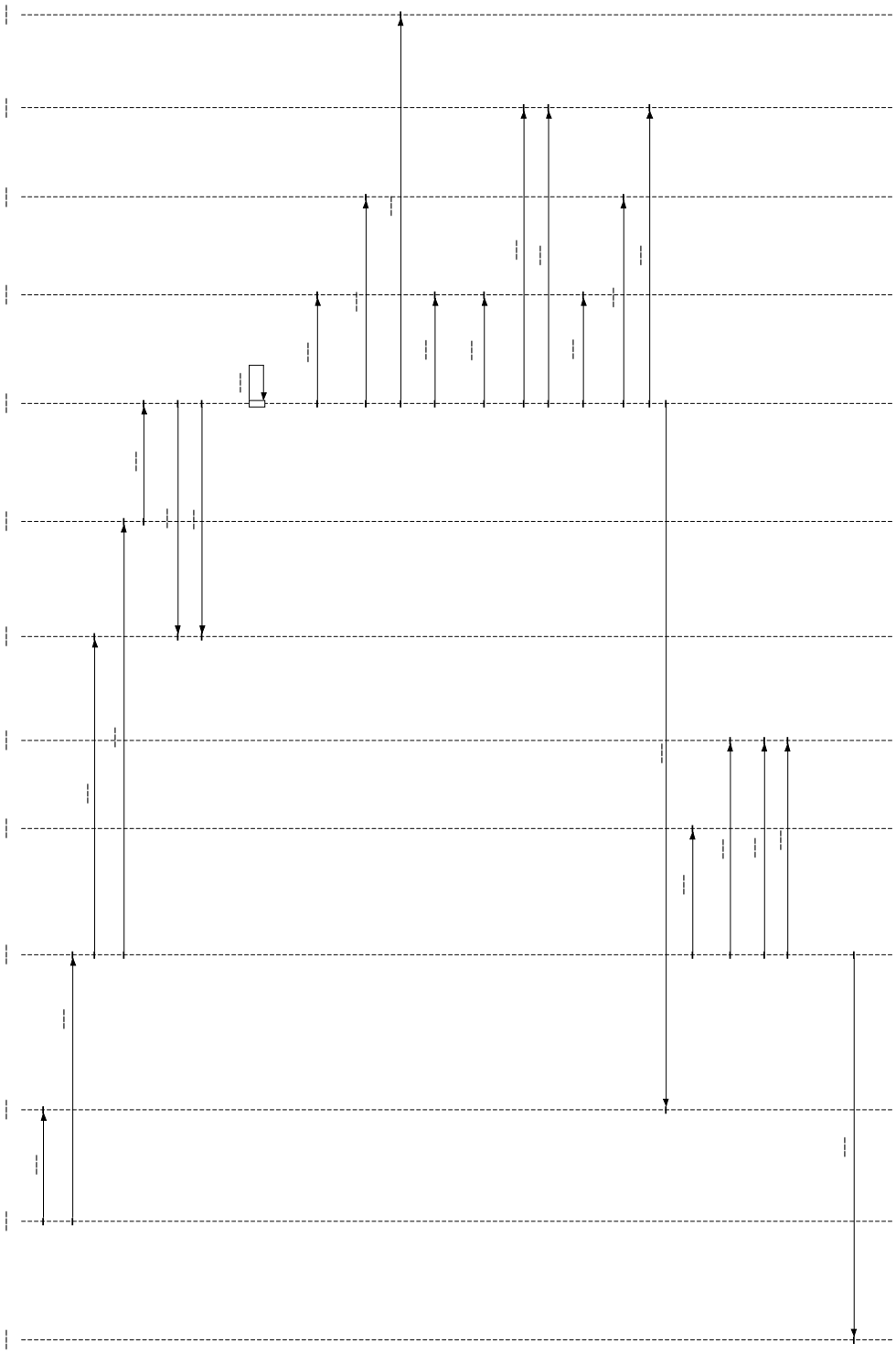


Figure 60 SDSRV Acquiring An ESDT

4.5.2 SDSRV Aster On Demand Process Request

4.5.2.1 Summary

This scenario shows how the SDSRV handles a client request for an ASTER Surface Emissivity granule requiring processing by the Planning and Data Processing Subsystem (PDPS). The service to produce the ASTER Surface Emissivity granule is advertised as part of the DsAsASTER03 ESDT. The processing options for the granule accompany the service request, and are passed to the Planning Subsystem (PLS), using the PlOnDemandPRNB public class. The PLS acknowledges receipt of the request, and the DsAsASTER03 class updates the PLS status to the server-side request class, DsShRequestReal, which returns status to the client. At the conclusion of processing, the PDPS performs the standard interactions with the SDSRV to insert the newly produced ASTER Surface Emissivity granule into the archive (see scenario SDSRV Inserting a Single ESDT). The DsAsASTER03 class gets the newly created granule from the PlOnDemandPRNB, and provides it to the server side object (see SDSRV Retrun Request Status scenario), which in turn provides the UR to the client software (see SDSRV Asynchronous Status update).

4.5.2.2 Assumptions

- Client software has already established a session with the SDSRV (see scenario SDSRV Client Connecting to a Data Server).
- Client software has already collected processing constraints from the user.

•Processing restraints are default parameters; the data server will permanently archive granules processed using default runtime parameters.

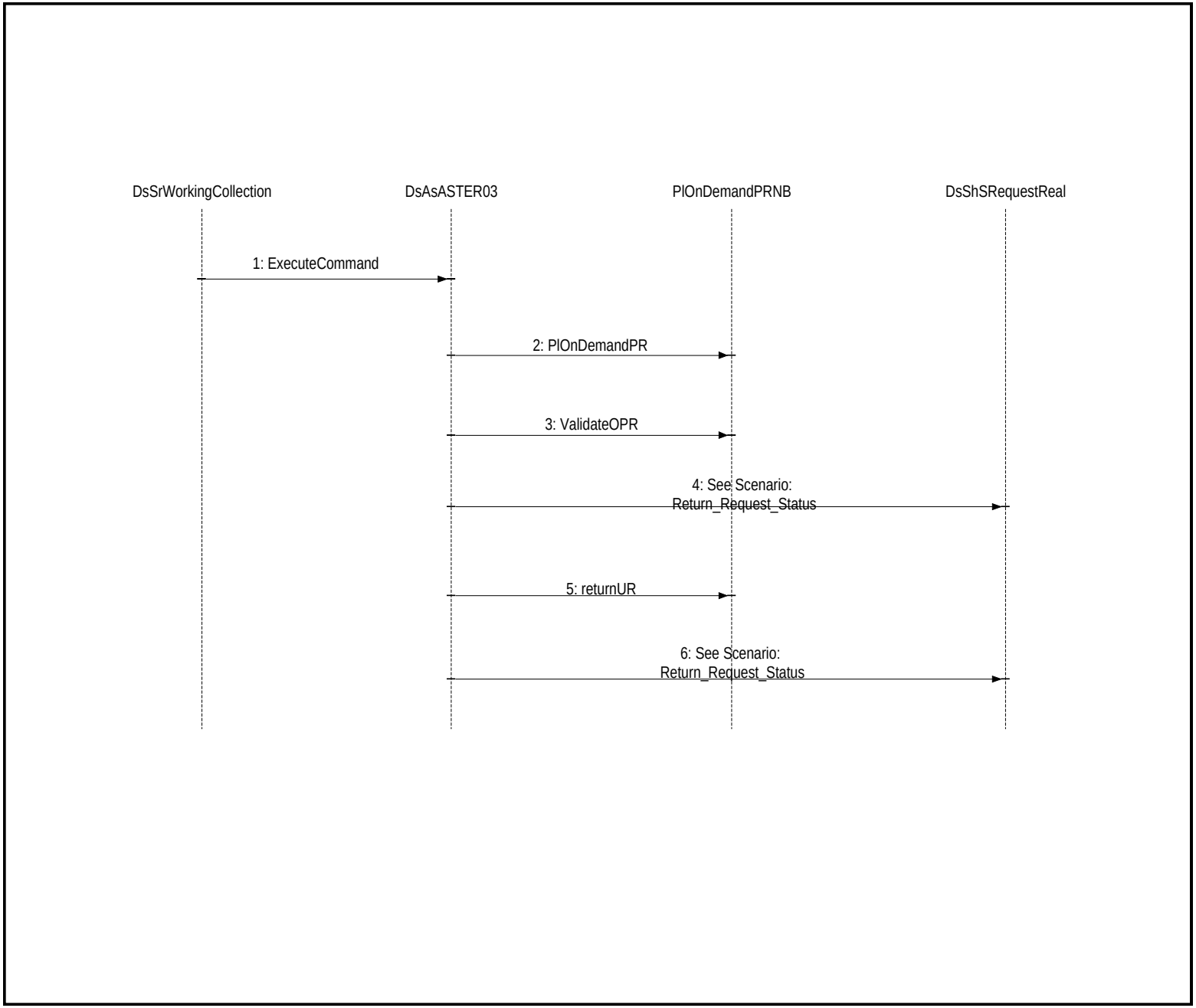


Figure 61. DSRV Aster On Demand Process Request

4.5.3 SDSRV Asynchronous Status Updates

4.5.3.1 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 DsShSRequest. DsShSRequest 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" DsShSRequest 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. Eventually, the client receives the DONE status and can obtain the results from the DsClRequest. Notice that the client never deals directly with the DsShSRequest , only with the DsClRequest which forwards calls to the DsShSRequest.

4.5.3.2 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

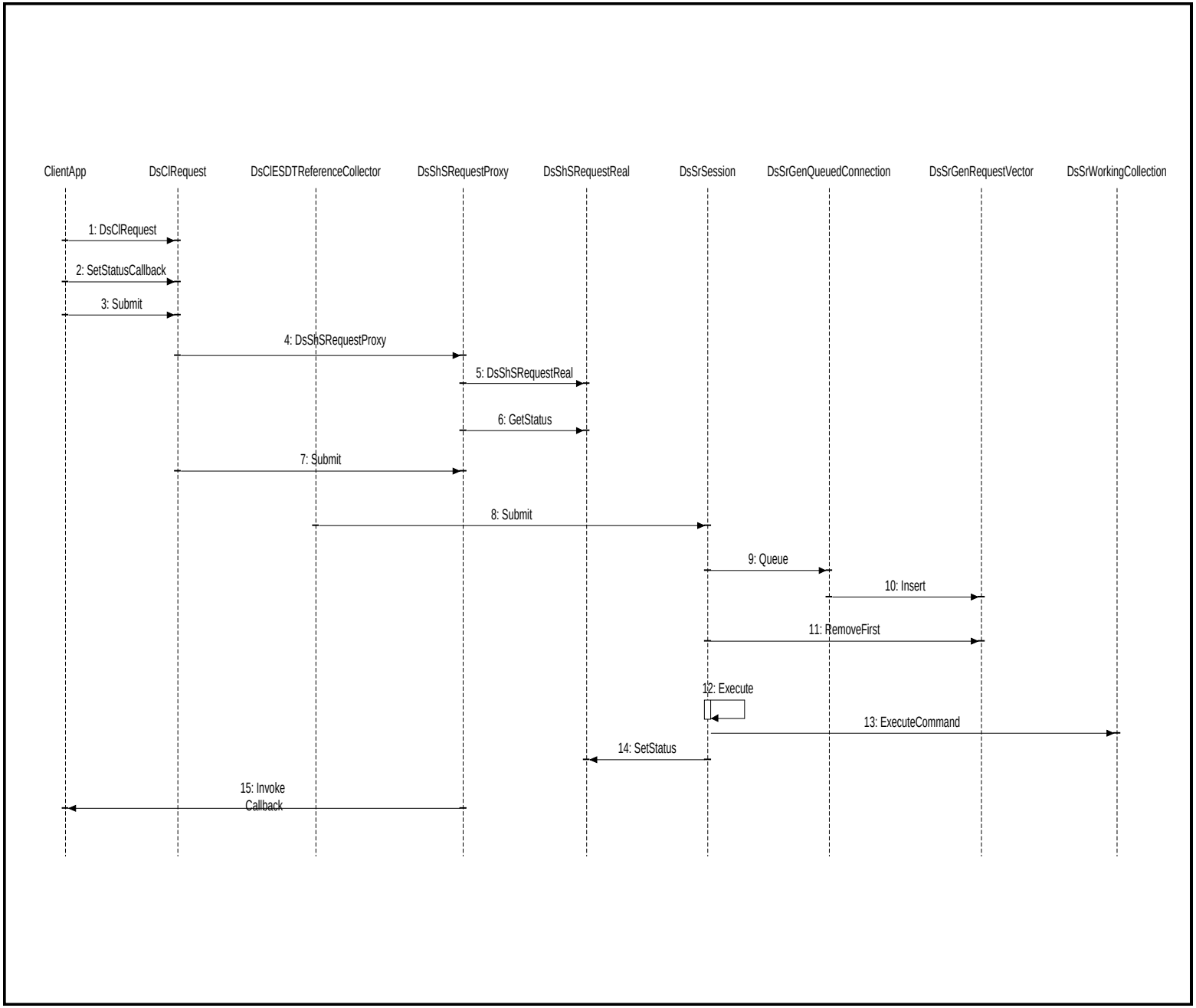


Figure 62 DSR V Asynchronous Status Updates

4.5.4 SDSRV Catalog Deleting A Metadata Entry

4.5.4.1 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 provided by Illustra rules or functions.

4.5.4.2 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.

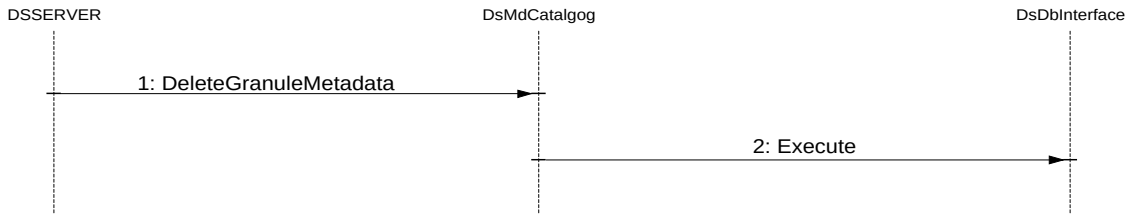


Figure 63. SDSR V Catalog Deleting A Metadata Entry

4.5.5 SDSRV Catalog Insert Collection Metadata

4.5.5.1 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 Illustra rules, functions, or dynamic SQL to insert the data.

4.5.5.2 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.

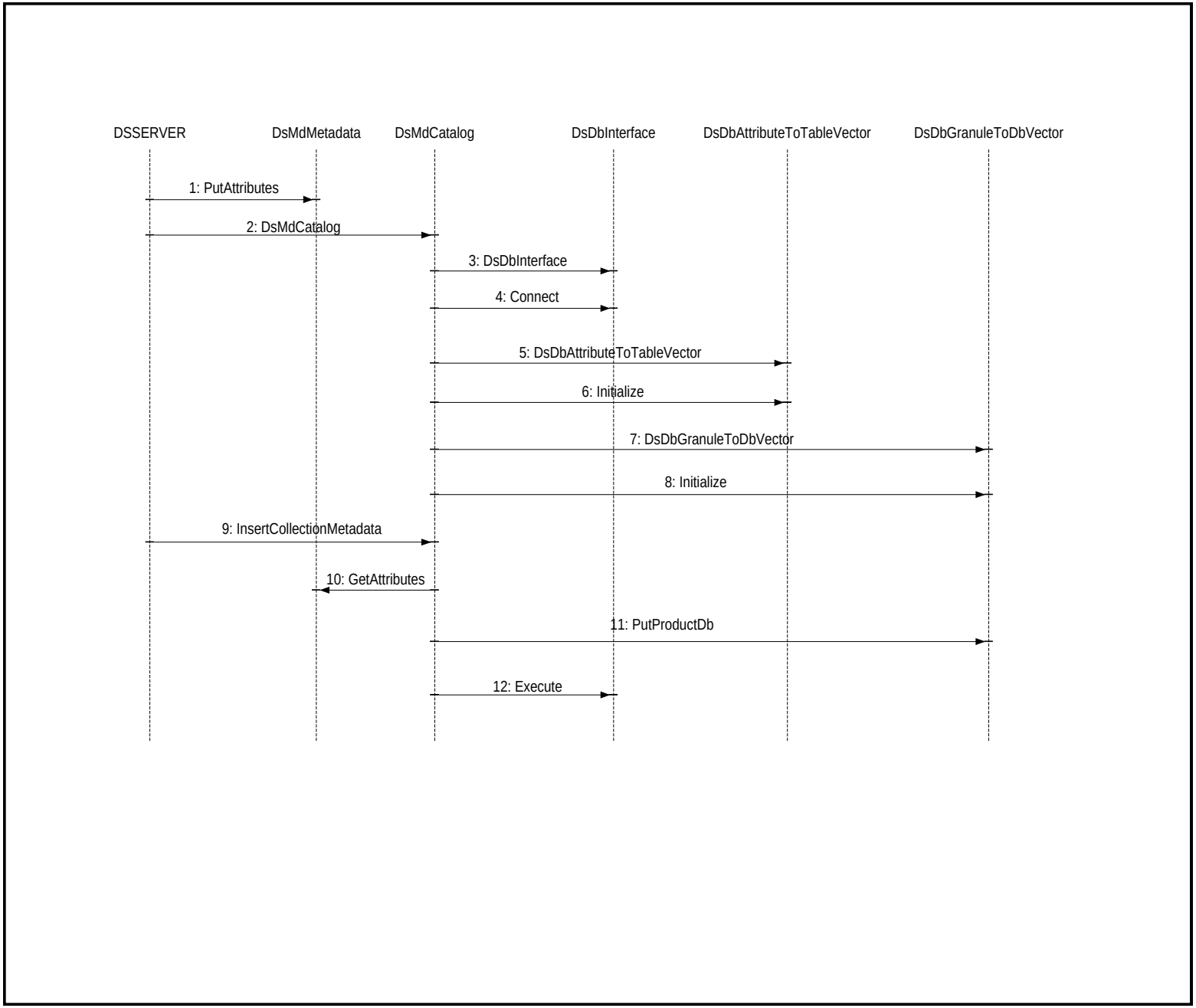


Figure 64. DSRV Catalog Insert Collection Metadata

4.5.6 SDSRV Catalog Insertion of Metadata

4.5.6.1 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 Illustra, executing Illustra commands, and mapping granules to the appropriate database.

4.5.6.2 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.

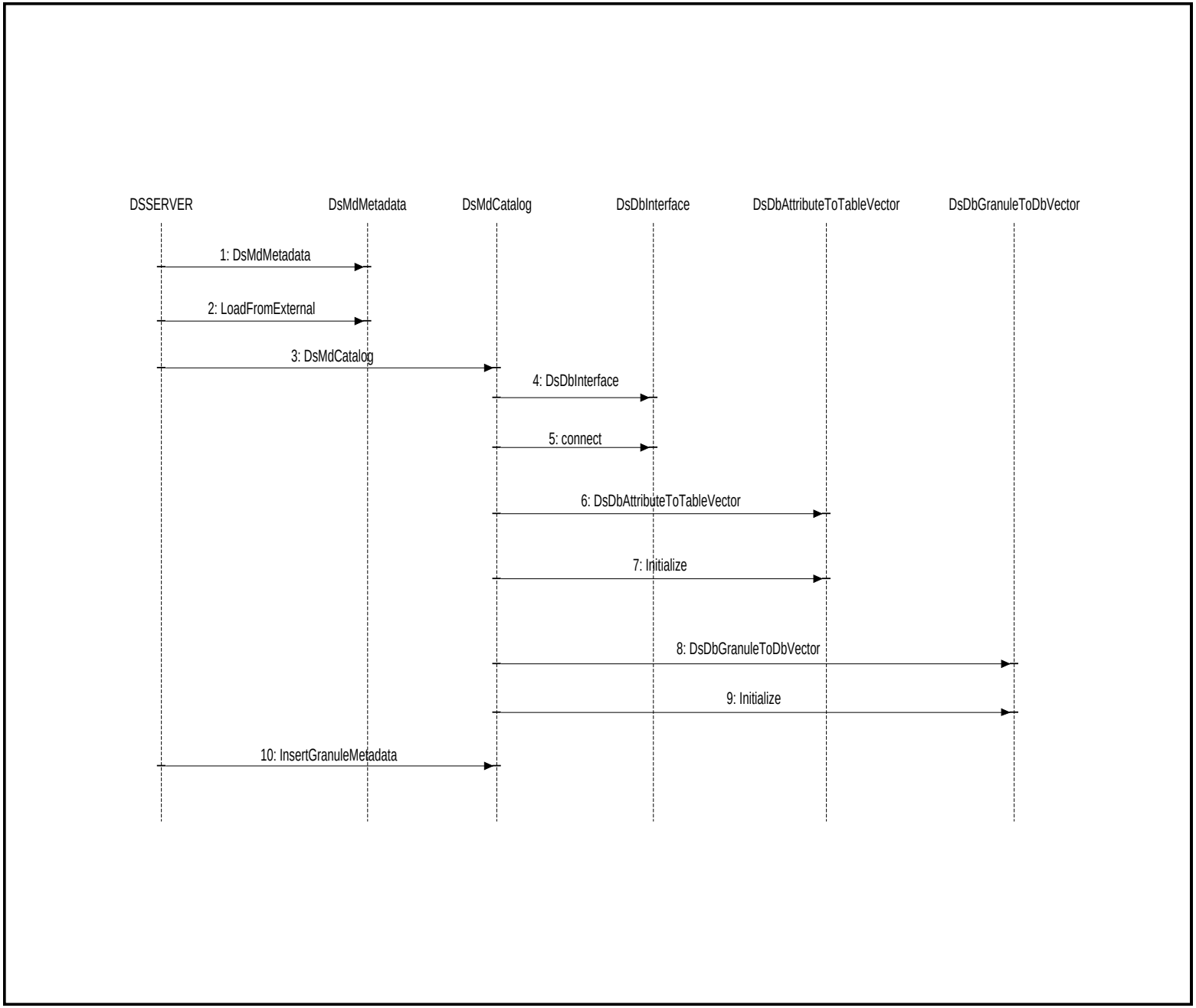


Figure 65. DSRV Catalog Insertion of Metadata

4.5.7 SDSRV Catalog Search

4.5.7.1 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 Illustra, executing Illustra 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.

4.5.7.2 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.

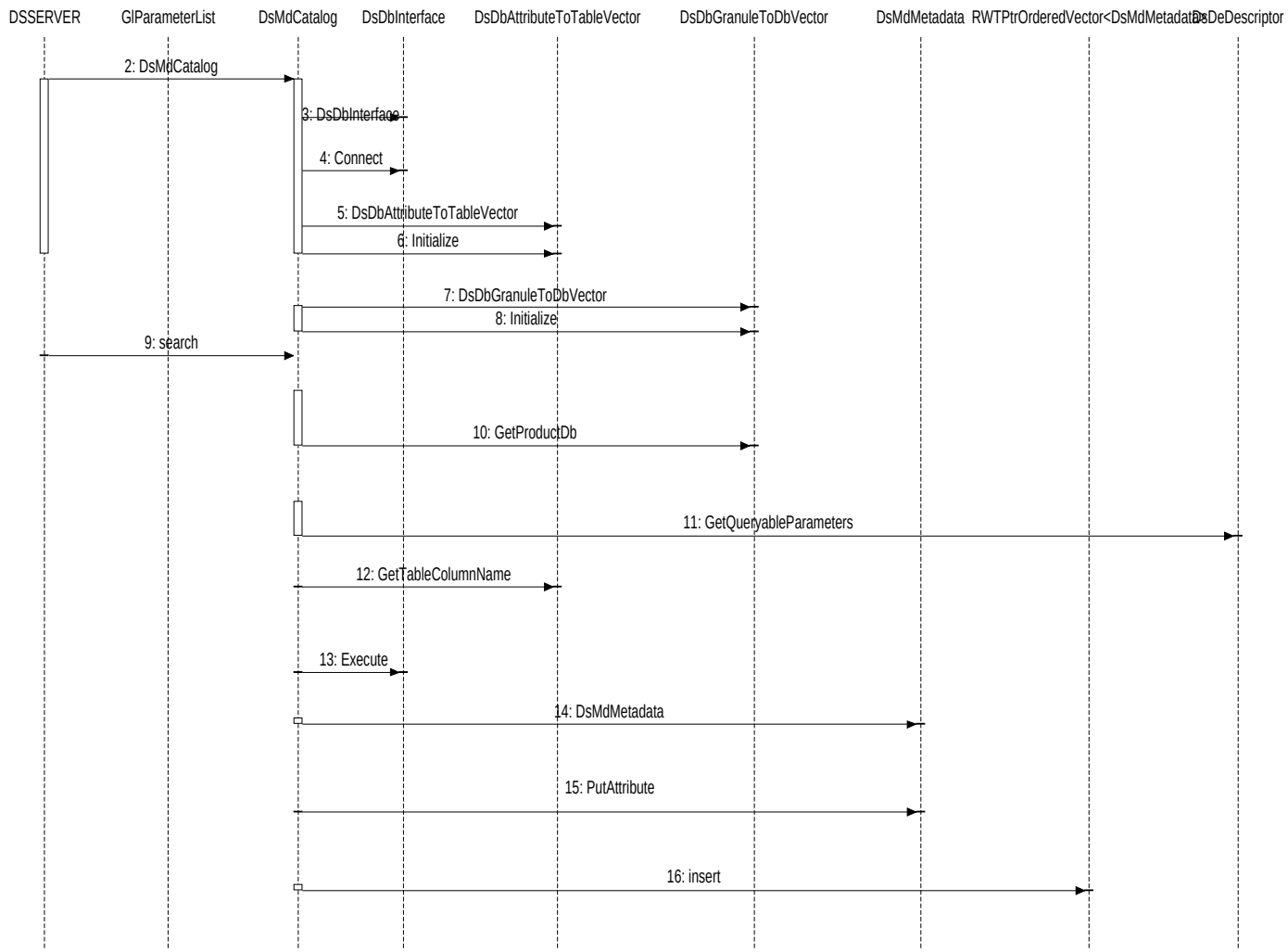


Figure 66 DSRV Catalog Search

4.5.8 SDSRV Catalog Updating Metadata

4.5.8.1 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.

4.5.8.2 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.

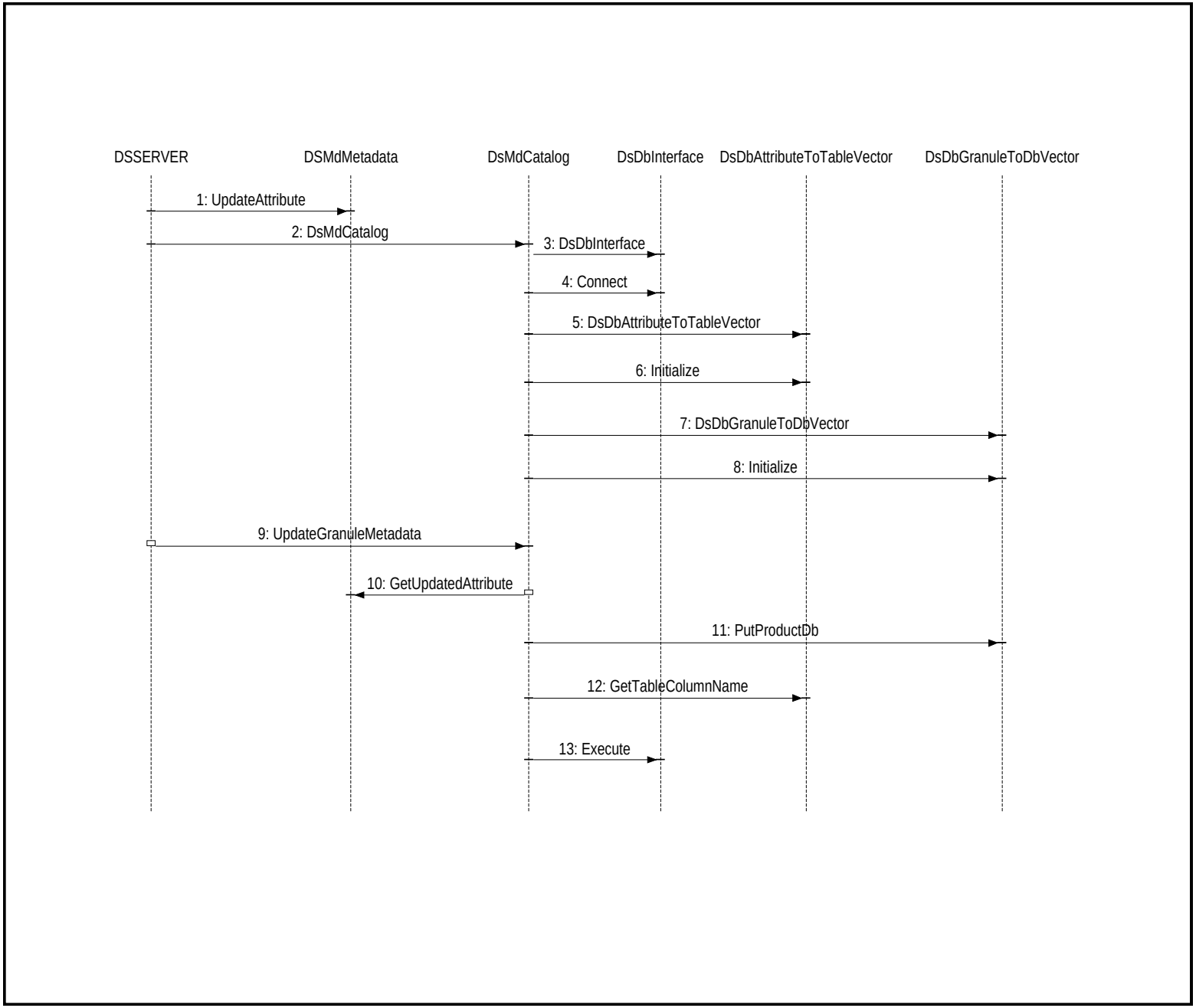


Figure 67. DSRV Catalog Updating Metadata

4.5.9 SDSRV Changing A Request Priority

4.5.9.1 Summary

This scenario shows how the priority is changed for requests that are currently being processed for a given ECS user, i.e., a p-thread has been spawned to handle an already submitted request.

The client application selects a request to set its priority to HIGH. The SetPriority method of the DsClRequest object is invoked. This, in turn, invokes the SetPriority method of the distributed object DsShSRequest to pass the priority change request to the server. DsShSRequestReal queries the MSS class MsAcUserProfile class to get information about the ECS user's group. This information provides "boundary" data (i.e., resource maximums and minimums) for ECS managed resources, such as CPU, for each class of ECS users. DsShSRequest then queries EcPfConfigfile to get the attribute list for the resources being managed by ECS. For this scenario, setting the request priority to HI implies that all p-threads being executed for the ECS user should have the managed resource boundaries bumped to the maximum limits set for the class to which the ECS user belongs. To perform this, DsShRequestReal invokes DCEPthread to reset the priorities for the requests belonging to the ECS user and to start the threads as appropriate.

4.5.9.2 Assumptions

•This scenario is for requests that are on the SDSRV queue or for active requests that spawn sub-threads; the priority of existing p-threads cannot be changed.

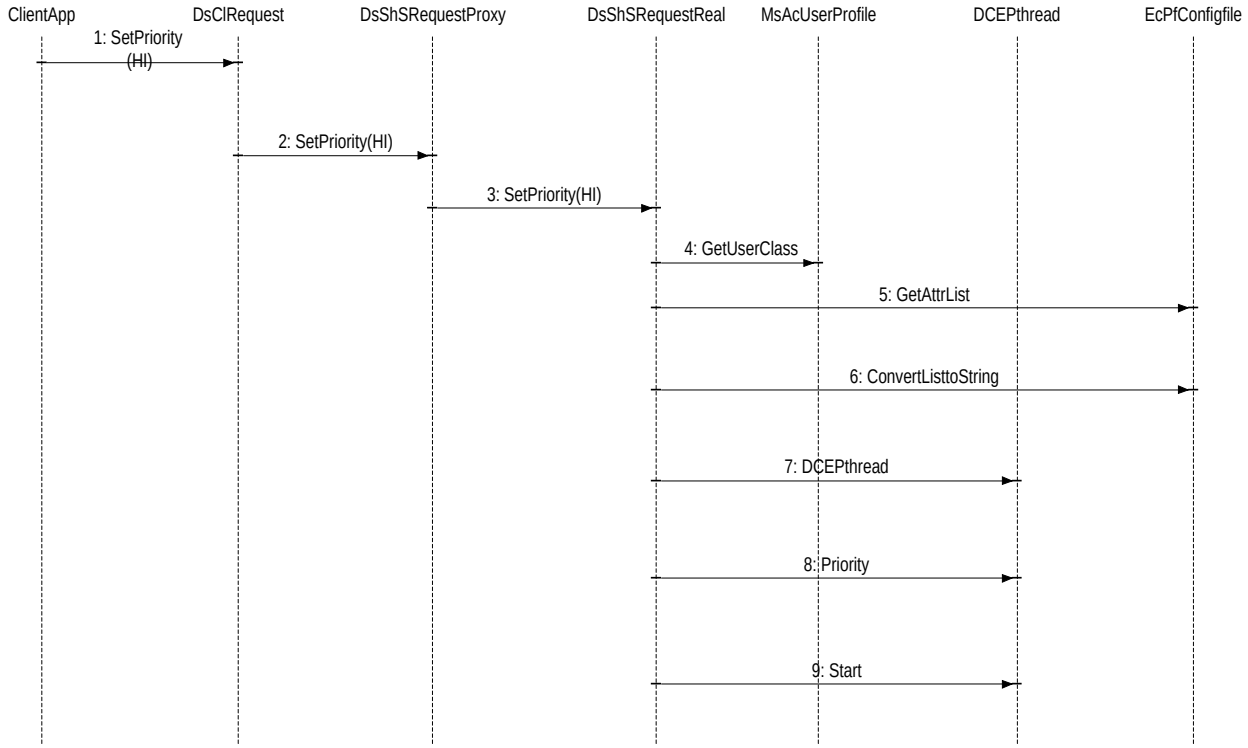


Figure 68: DSR V Changing A Request Priority

4.5.10 SDSRV Client Browsing

4.5.10.1 Summary

This scenario shows how the client classes are used to retrieve a browse image 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 contains 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.

4.5.10.2 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

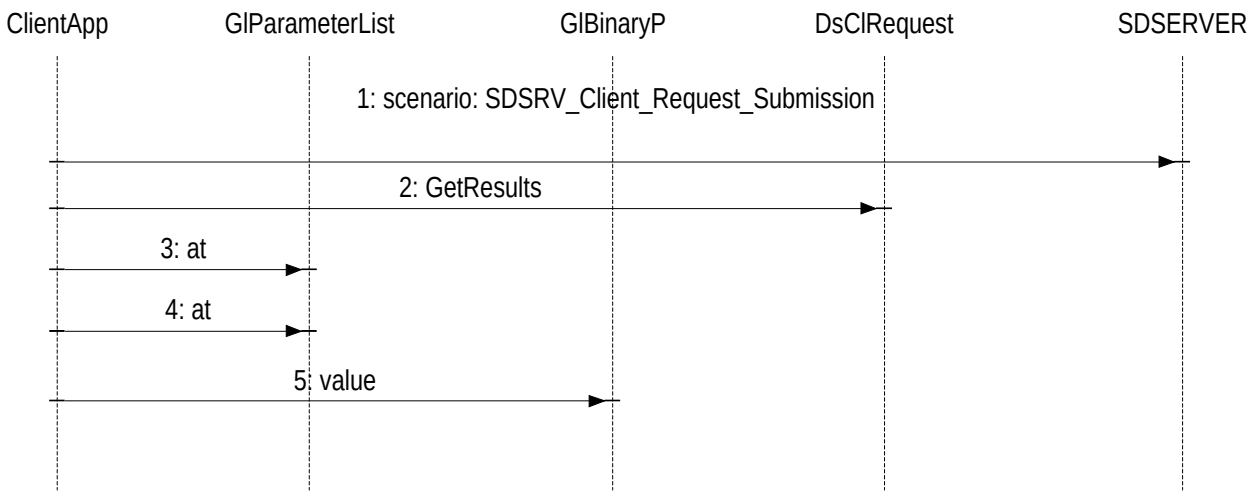


Figure 69 SDSRV Client Browsing

4.5.11 SDSRV Client Connecting to a Data Server

4.5.11.1 Summary

The DsClESDTReferenceCollector object is a distributed object on the client side. When the client software creates the DsSeConnectionMaker on the server side, the DsSrSession object is created on the server side. This is done via the CSS Distributed Object Framework. 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 access to common 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 initialized, 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.

4.5.11.2 Assumptions

- Client software knows the dataserver to which it needs 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 to SDSRV

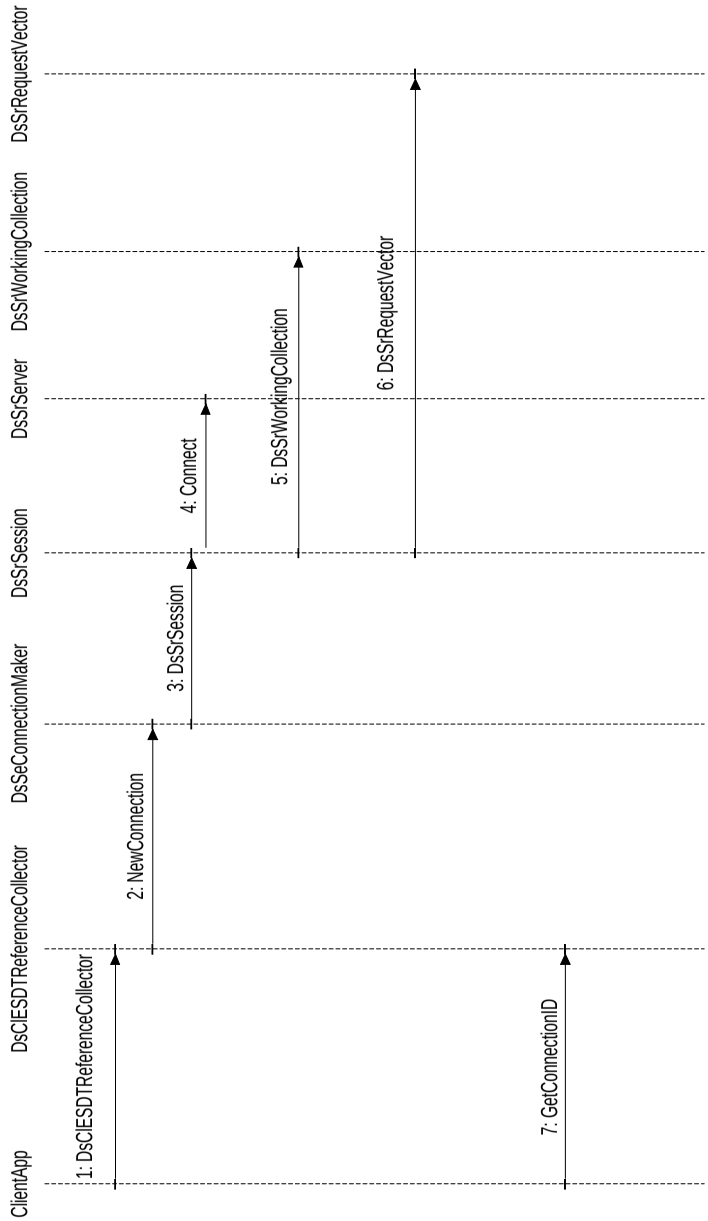


Figure 70. GDSR V Client Connecting to a Data Server

4.5.12 SDSRV Client Request Submission

4.5.12.1 Summary

This scenario shows how the client classes handle a request. The SDSRV Asynchronous Status Updates scenario shows the details of how the request reports status updates to the client; this scenario shows the steps used to submit the request 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 destroy the request and command objects.

4.5.12.2 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

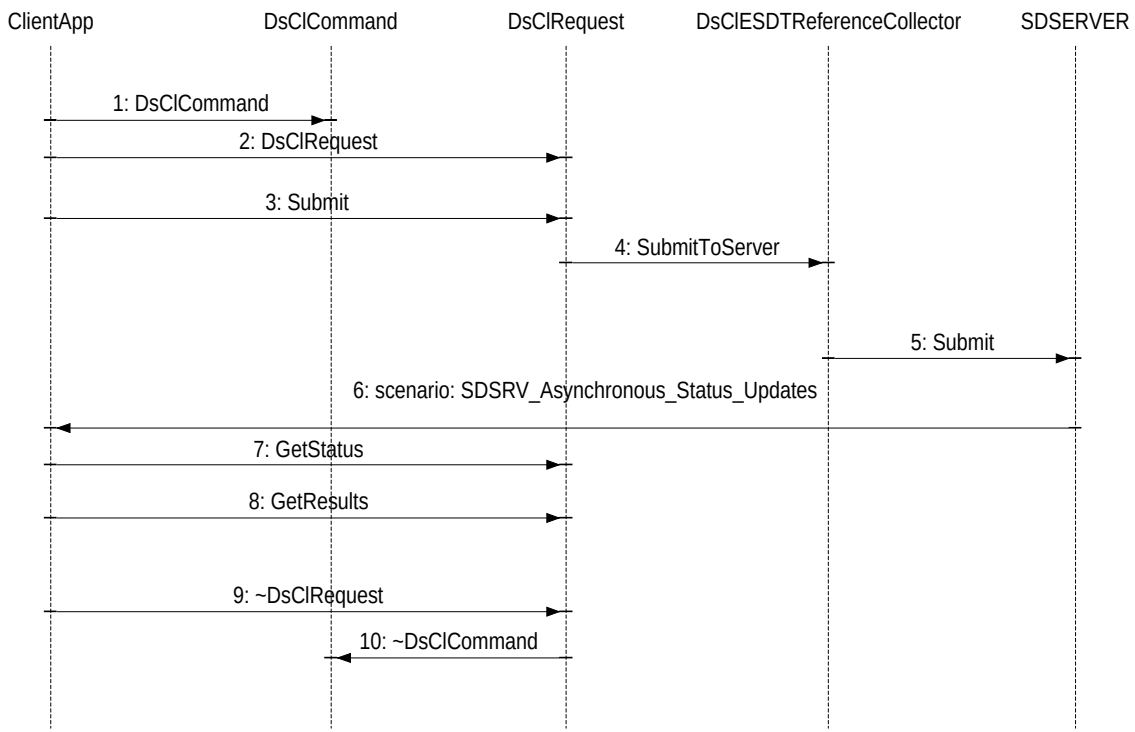


Figure 71 SDSRV Client Request Submission

4.5.13 SDSRV Client Searching

4.5.13.1 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 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. (See SDSRV Asynch status updates.)

The constraints are passed to DsClQuery, via a GlParameterList containing the names and corresponding values of search parameters. 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 Convert to Command 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). Each "add" command transfers an ESDT reference from the server side DsSrWorkingCollection to the DsClESDTReferenceCollector on the client side. (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 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.

4.5.13.2 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

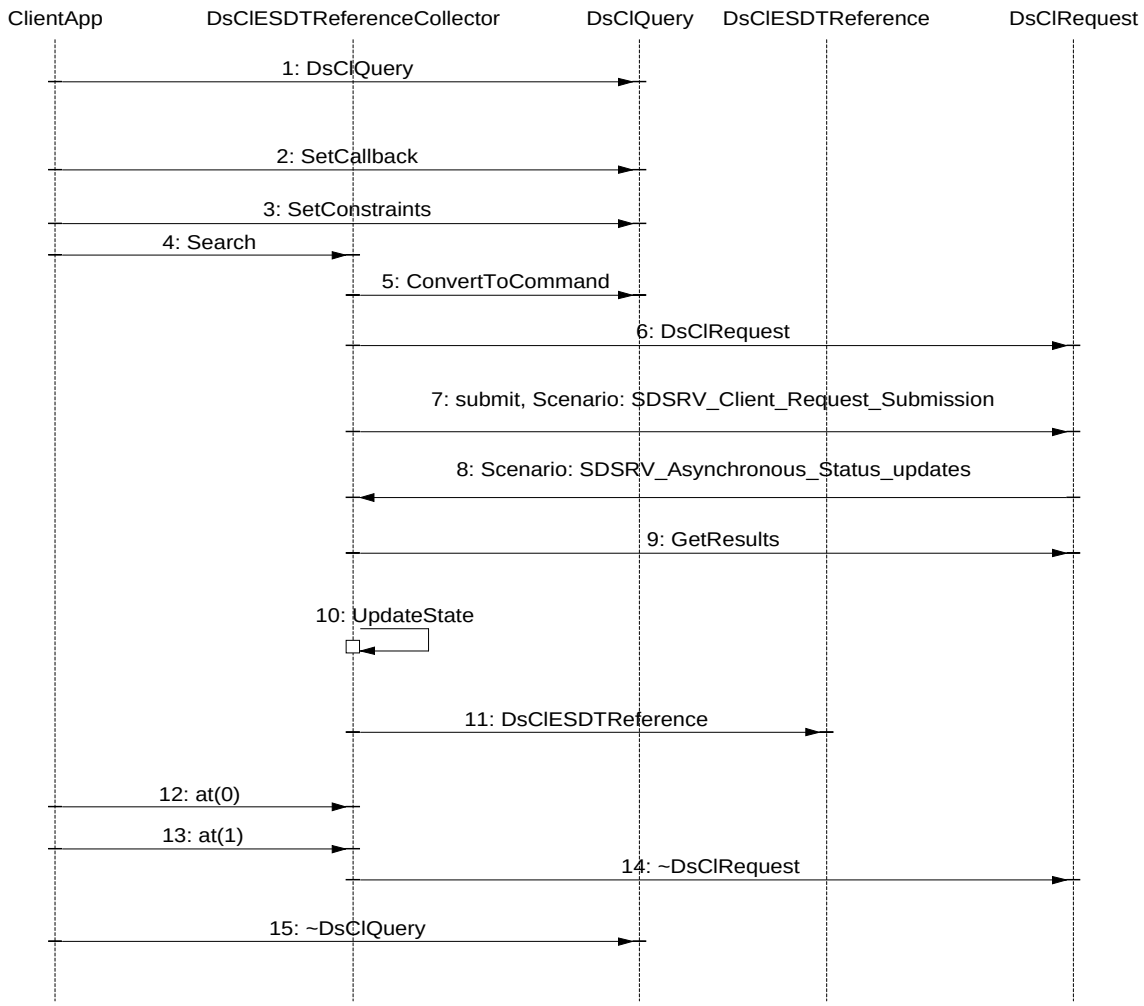


Figure 72. SDSRV Client Searching

4.5.14 SDSRV Deleting A Queued Request

4.5.14.1 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 and removed from the request vector on the server side. 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.

4.5.14.2 Assumptions

- the client has all necessary permissions
- the request is queued on the server

•all steps execute successfully

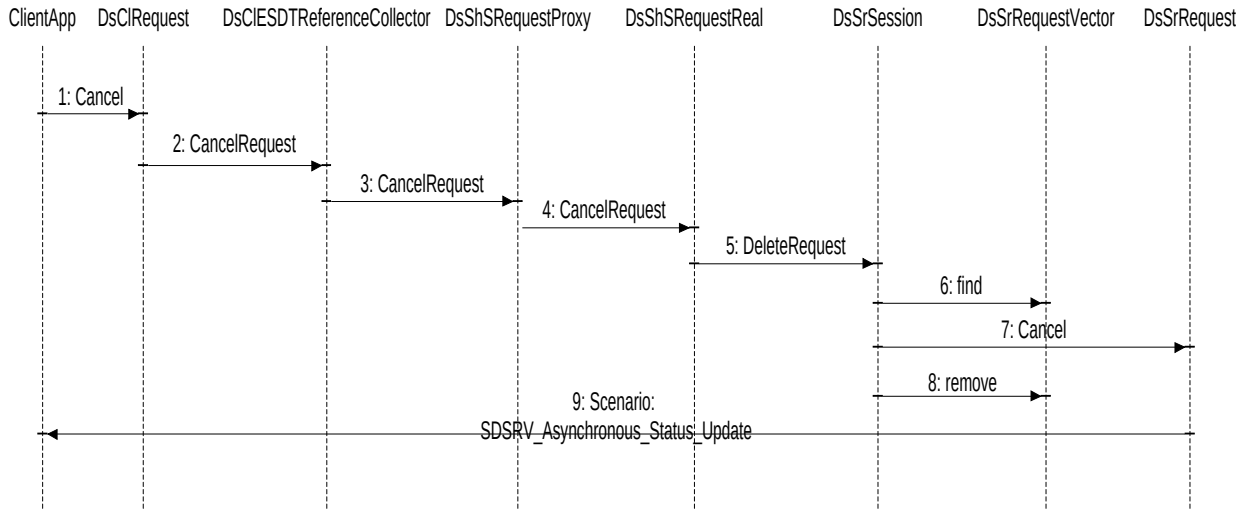


Figure 730 SDSRV Deleting A Queued Request

4.5.15 SDSRV Inserting Composite ESDT

4.5.15.1 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.

4.5.15.2 Assumptions

- Ingest supplies a file containing valid 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

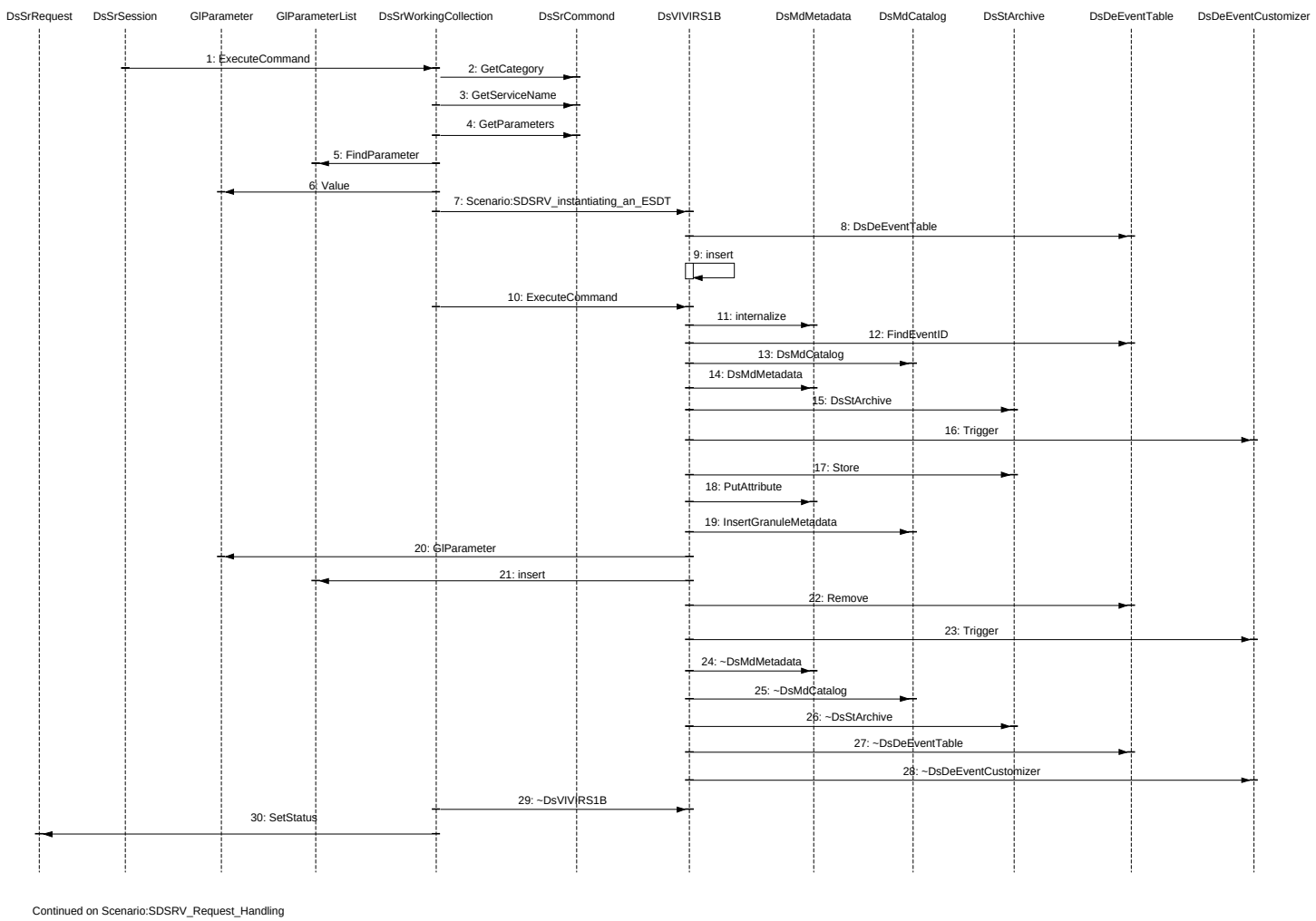


Figure 74. SDSRV Inserting Composite ESDT

4.5.16 SDSRV Inserting New ESDT

4.5.16.1 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.

4.5.16.2 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.

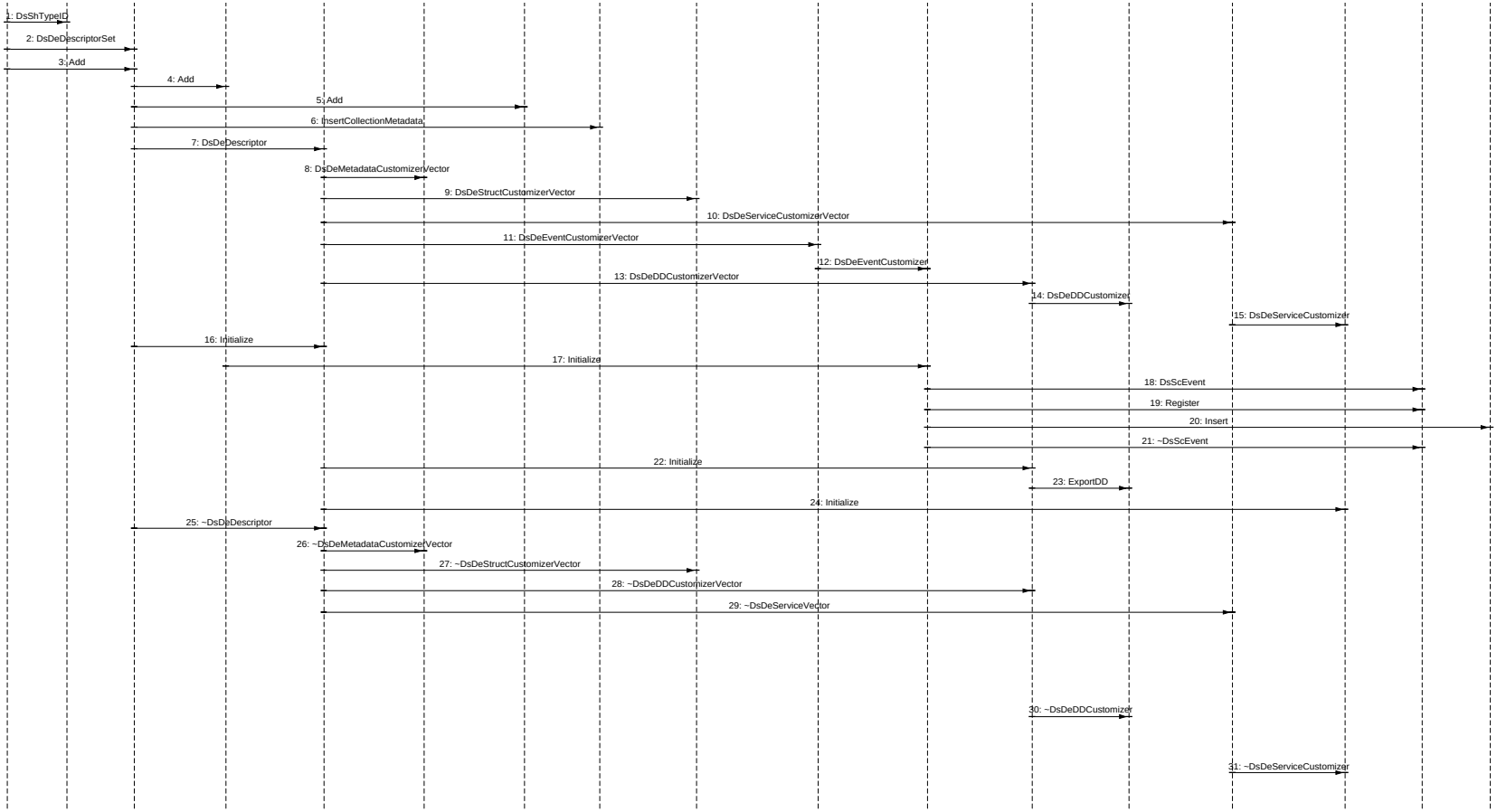


Figure 75. SDSRV Inserting New ESDT

4.5.17 SDSRV Inserting Single ESDT

4.5.17.1 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.

4.5.17.2 Assumptions

- Ingest supplies a file containing valid 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

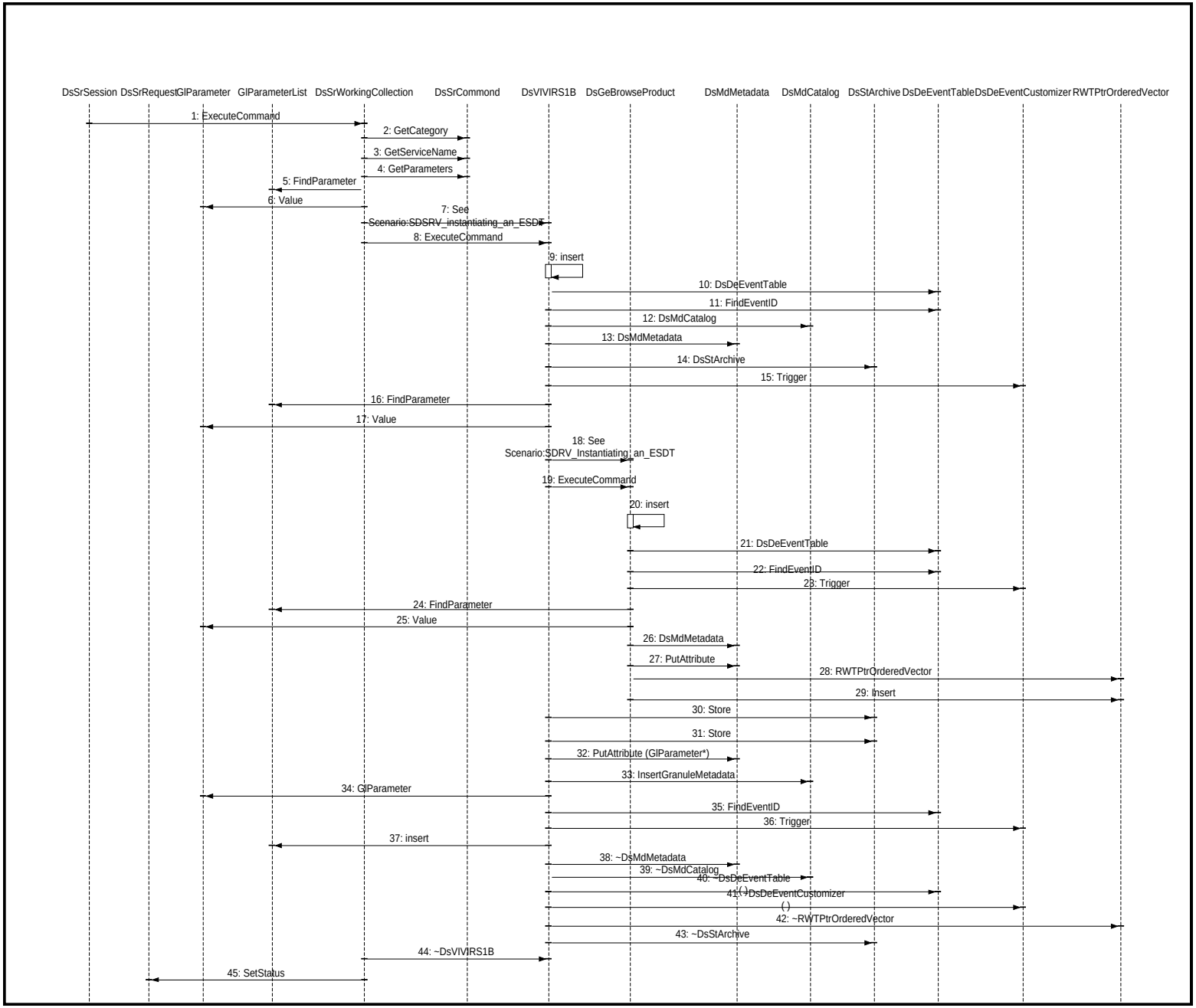


Figure 76 SDSRV Inserting Single ESDT

4.5.18 SDSRV Instantiating An ESDT

4.5.18.1 Summary

The scenario begins with the working collection determining what data type is to be instantiated. The ESDTwrapper makes a call to the DsGeDynamicLibraryObject, including in that call the data type required. Using appropriate lookup tables the relevant dynamic link library (DLL) 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.

4.5.18.2 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.

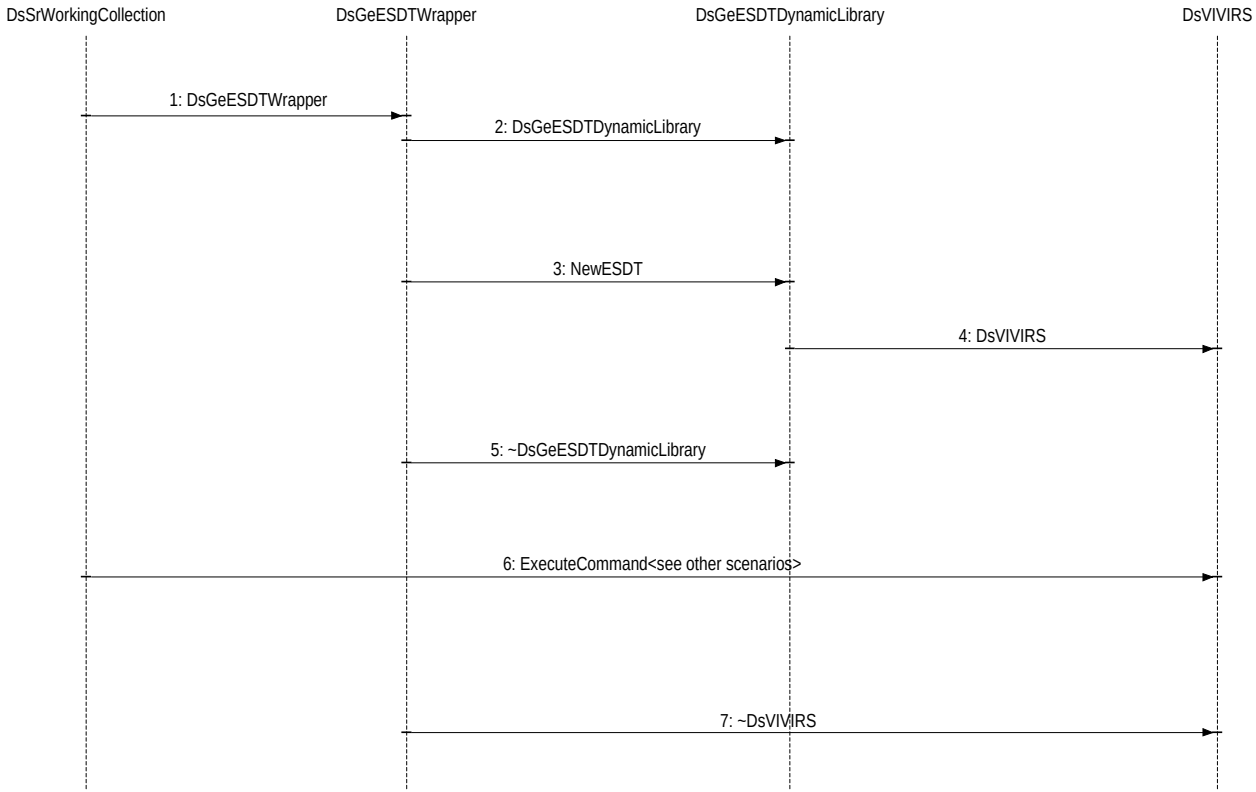


Figure 77 DSR V Instantiating An ESDT

4.5.19 SDSRV Multi State Granule

4.5.19.1 Summary

This scenario shows how a MISR05 granule awaiting QA, and visible only to a select group of ECS users, undergoes a successful quality assurance update and becomes visible to the general science user community. This is accomplished by creating two ESDTs, each representing the granule in a different state, i.e., “awaiting QA” and “QA performed”. Ths state transition is accomplsihed by changing the metadata for the granule from its initial ESDT, MISR05, TO ITS NEW ESDT, MISR05 QA. The service to perform a QA update on a MISR05 granule is created as part of the DsMsMISR05 ESDT.

The quality assurance update parameters for the granule package accompany the service request and are saved by DsMdMetadata as an ODL stream, along with the other metadata belonging to the MISR05 granule. DsMsMISR05 QA then instantiates and inserts a new granule into the metadata repository using the ODL stream with the new QA values. (see the scenarios SDSRV Instantiating an ESDT and SDSRV Inserting a Single ESDT). The old (i.e., non-QA'd) granule instance is then deleted from the catalog using the MdCatalog class. The DsMISR05 class then updates the status of the QA update to the server-side request class, DsShRequestReal, where it passes the status to the client. Now the granule package is visible to the general science user community using the services provided by the MISR05 QA ESDT.

4.5.19.2 Assumptions

- The MISR05 granule package has been inserted and is awaiting QA.
- Client software has already established a session with SDSRV (see scenario SDSRV Client Connecting to Data Server).
- Client software has already collected the quality assurance update parameters from the user.
- Only the granule package MISR05 metadata is affected by the QA update; the archived files remain untouched as part of the scenario.

•This scenario covers QA of a MISR05 granule package, but the same approach would be used for QA of any other product granule package.

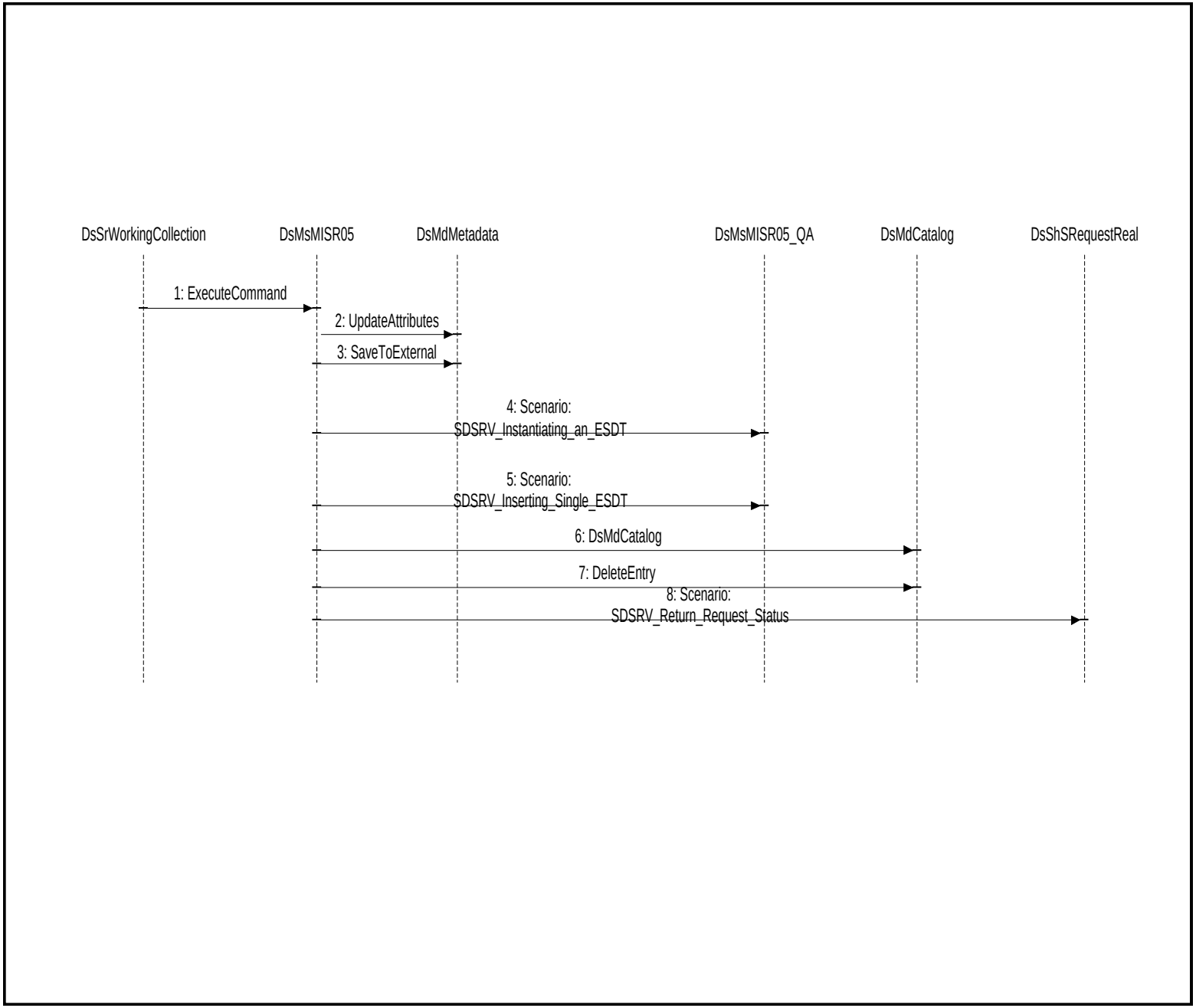


Figure 78. SDSRV Multi State Granule

4.5.20 SDSRV Op View Queued Requests

4.5.20.1 Summary

This scenario shows how an SDSRV operator's GUI gets the necessary information about currently queued requests from a Science Data Server.

4.5.20.2 Assumptions

- GUI knows which SDSRV the operator is interested in.

•GUI formats the text representation of the queued requests in a scrollable text widget.

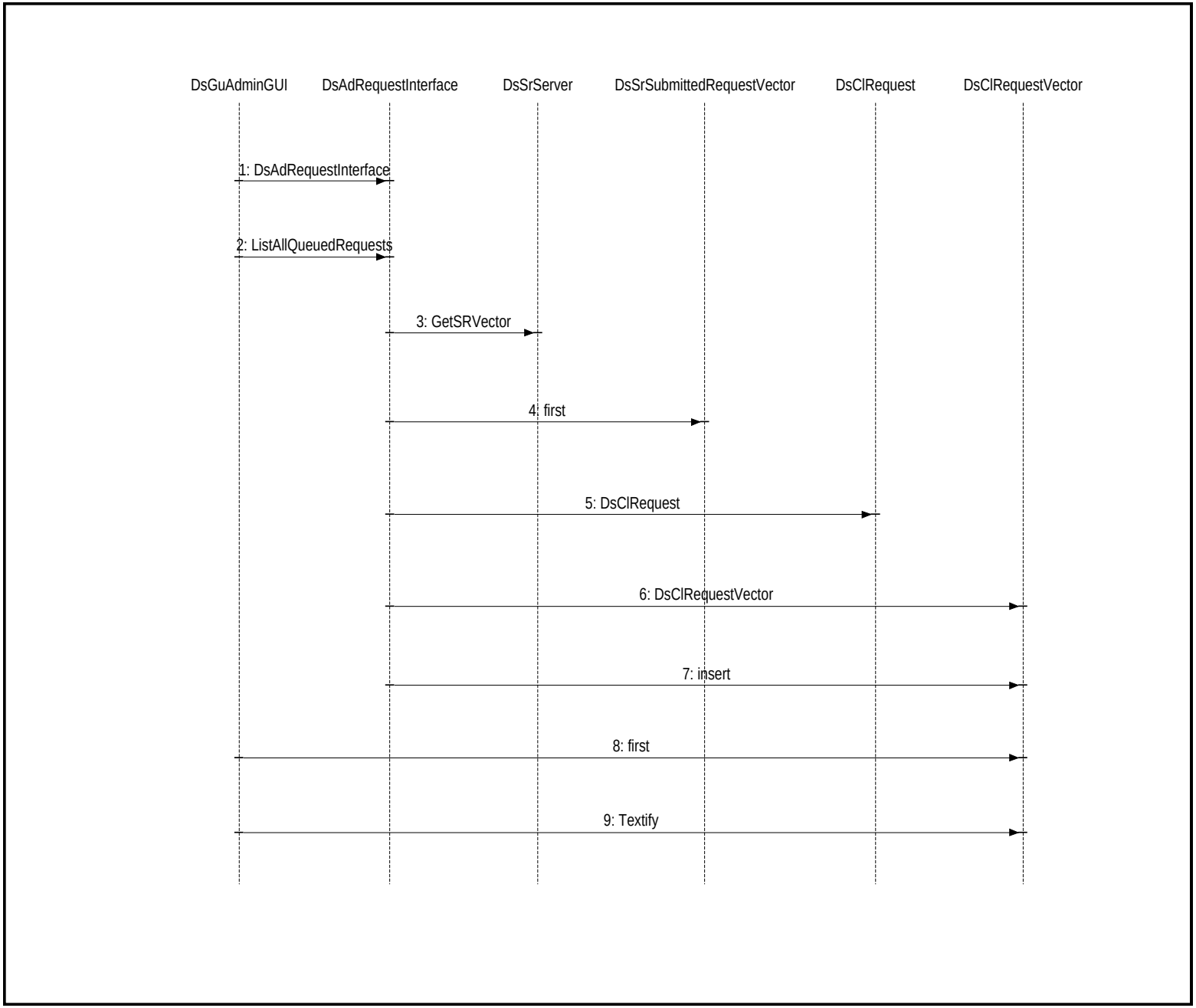


Figure 798 DSR V Op View Queued Requests

4.5.21 SDSRV Server Handling A Browse Request

4.5.21.1 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, in this case a byte stream that contains the browse information, are added to the request results, and the remainder of the SDSRV Server Request Handling scenario is finished.

4.5.21.2 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

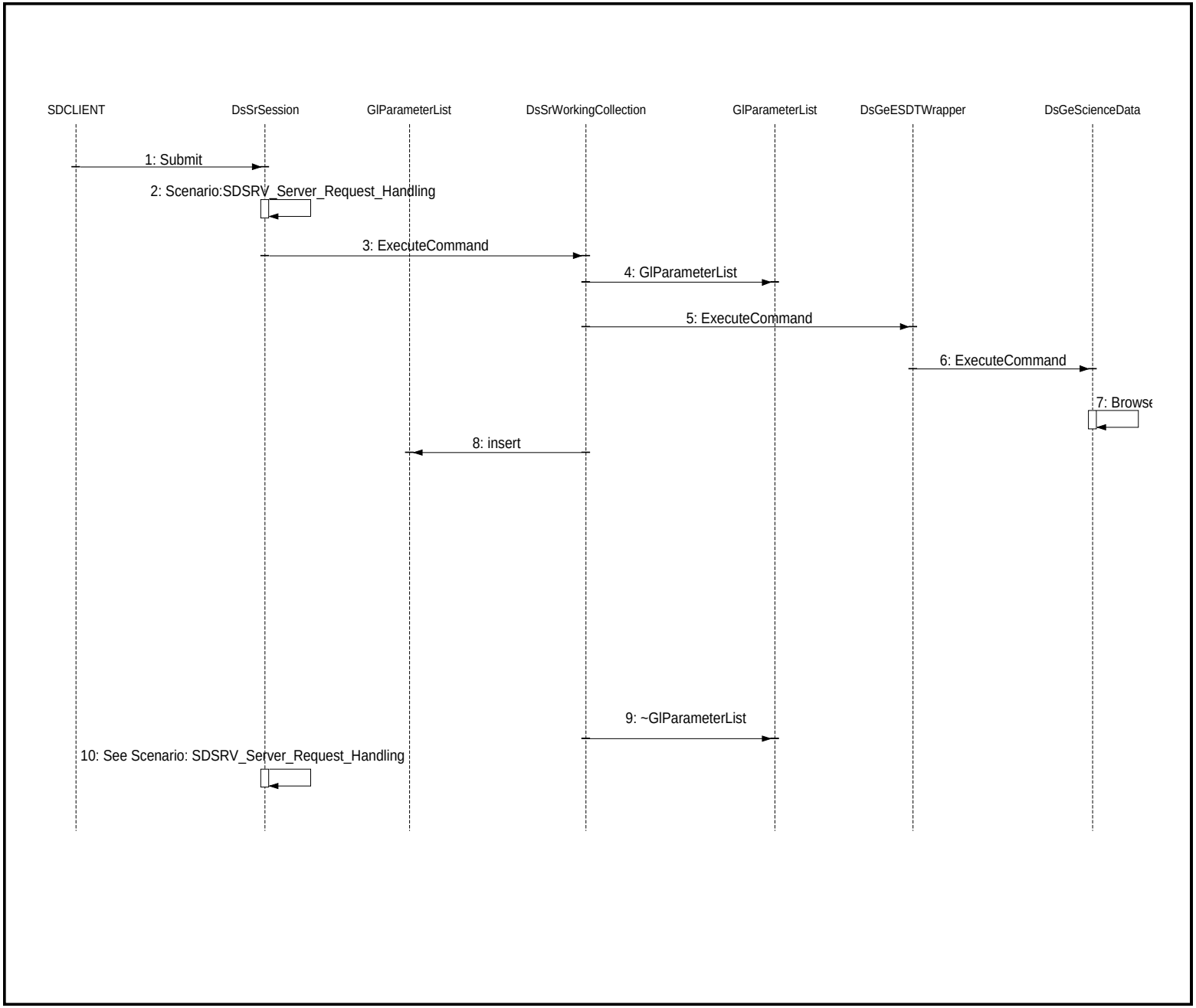


Figure 80 SDSRV Server Handling A Browse Request

4.5.22 SDSRV Server Handling A Search Request

4.5.22.1 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 GlParameterList 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.

4.5.22.2 Assumptions

- All corresponding client-side proxies have been established
- DOF Infrastructure 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

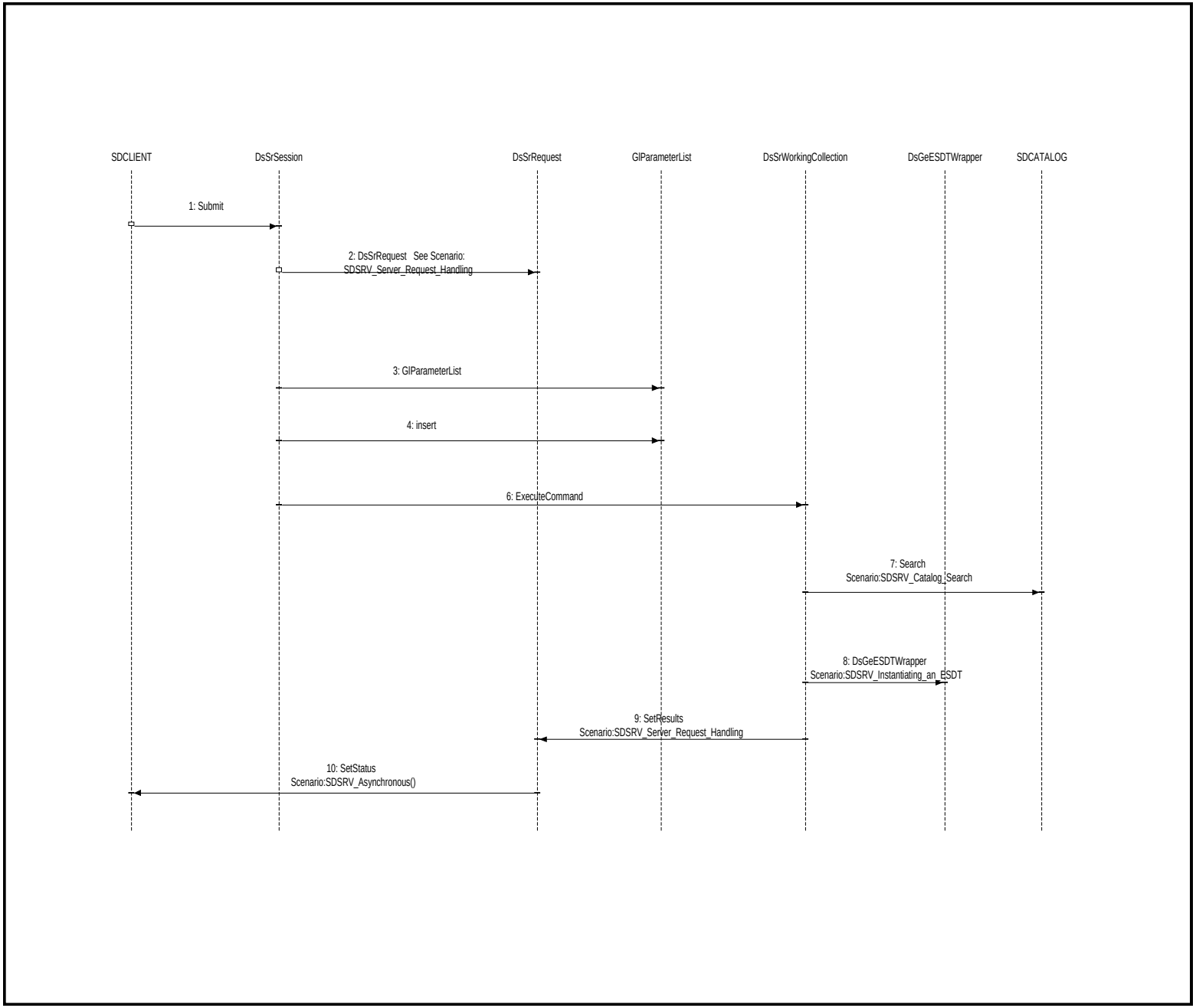


Figure 81 SDSRV Server Handling A Search Request

4.5.23 SDSRV Server Request Handling

4.5.23.1 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) DsShSRequestProxy and the unique ID of the DsShSRequestReal is sent to DsSrSession. DsSrSession uses this ID to find the DsShSRequestReal 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.

4.5.23.2 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

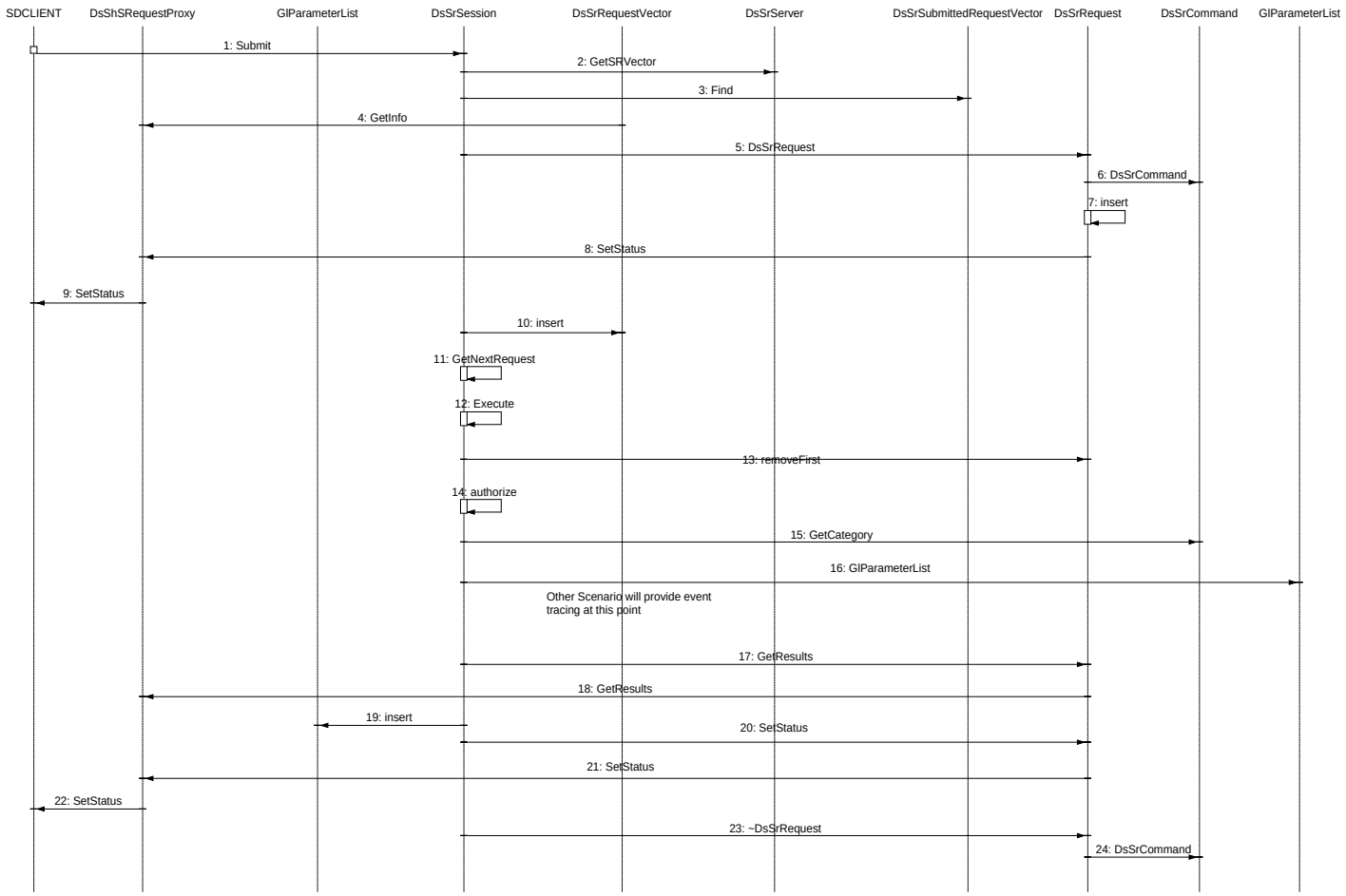


Figure 82. SDSRV Server Request Handling

4.5.24 SDSRV Startup Of A Science Data server

4.5.24.1 Summary

This scenario shows the steps required to startup a data server. The PFManagedProcess 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 SDSRV Inserting A New Data Type.

4.5.24.2 Assumptions

None

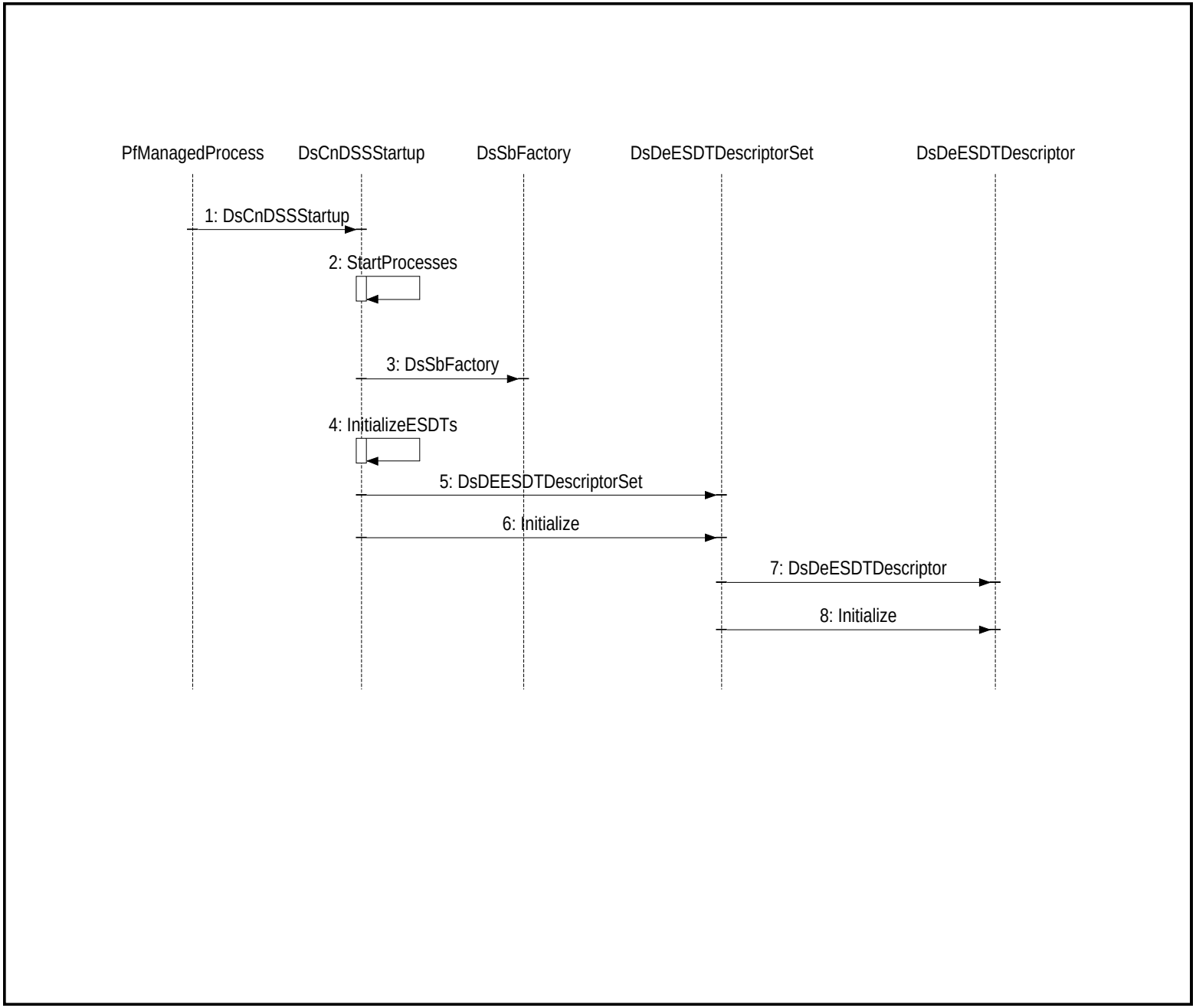


Figure 83. DSR V Startup Of A Science Data server

4.5.25 SDSRV Subsetting an ESDT

4.5.25.1 Summary

This scenario shows how a MISR05 product is spatially subsetted and the UR for the subsetted file, which exists for the duration of the ECS user session, is returned to the SDSRV client. The service to perform spatial subsetting is created as part of the MISR05 ESDT.

The DsSrWorkingCollection class receives the command and uses the DsSrCommand class to get the service name (SpatialSubset) and the parameters necessary to implement the service. The service name and its parameters are validated by the DsDeDescriptor class. Now the SDSRV can begin executing the SpatialSubset service.

First, the SDSRV performs the "preparations" for creating the subsetted file. DsMsMISR05 determines how much space needs to be allocated for the files (the orginal plus the subsetted file) in Working Storage. The STMGT class DsStStagingDisk then allocates the necessary disk space and creates the handle for the file that will hold the subsetted product. DsMsMISR05 invokes the DsMdMetadata class to get the metadata granule attributes for the archived file that is to be subsetted. Next, the SDSRV connects to STMGT, sends the file parameters to DsStFileParameters, and uses DsStArchive to retrieve the archived file from the FSMS and place it into working storage. SDSRV then disconnects from STMGT.

Because MISR05 is a gridded product, the SpatialSubset service for the MISRO5 ESDT invokes DsCsGrid to begin preparations for a gridded product file extract. Using the DsCsCSDTFactory class, a CSDT implementation is created. Because the product is in HDF-EOS format, DsCsCSDTFactory invokes the DsCsHDF-EOSGrid to actually handle the file extraction for the subset. DsCsHDF-EOS invokes low level HDF-EOS API calls to perform the extraction. After HDF-EOS API call (GDopen) to open the archive file that is in working storage, DsMsMIS05 is invoked to get the spatial subsetting parameters. DsMsMIS05 iterates through invocations (GlParameterList) to get the list of coordinates that define the subsetting area. DsCsGrid defines the extract area (using the parameter list) to be used by DsCsHDF-EOSGrid. After the extract area is defined, DsCsHDF-EOSGrid executes a series of HDF-EOS API calls to perform the extraction from the archived file into the new file. This extraction includes the granule metadata with updates as are appropriate for the extract area.

After the subsetted file has been created, the SDSRV prepares to complete the command execution. DsMsMISR05 invokes GlUR to create a new UR for the subsetted file. The metadata attributes for the newly subsetted file are updated using DsMdMetadata. Finally, the UR is added to the GlParameterList and the status is made available to the client (see scenario Returning Request Status).

4.5.25.2 Assumptions

- A connection has already been established to the SDSRV

- A working collection already exists
- The working collection contains at least one granule as a result of a valid search
- The "Spatial Subset" service is advertised as available for the MISR05 granule
- The granule science data is formatted according to the HDF-EOS grid specifications

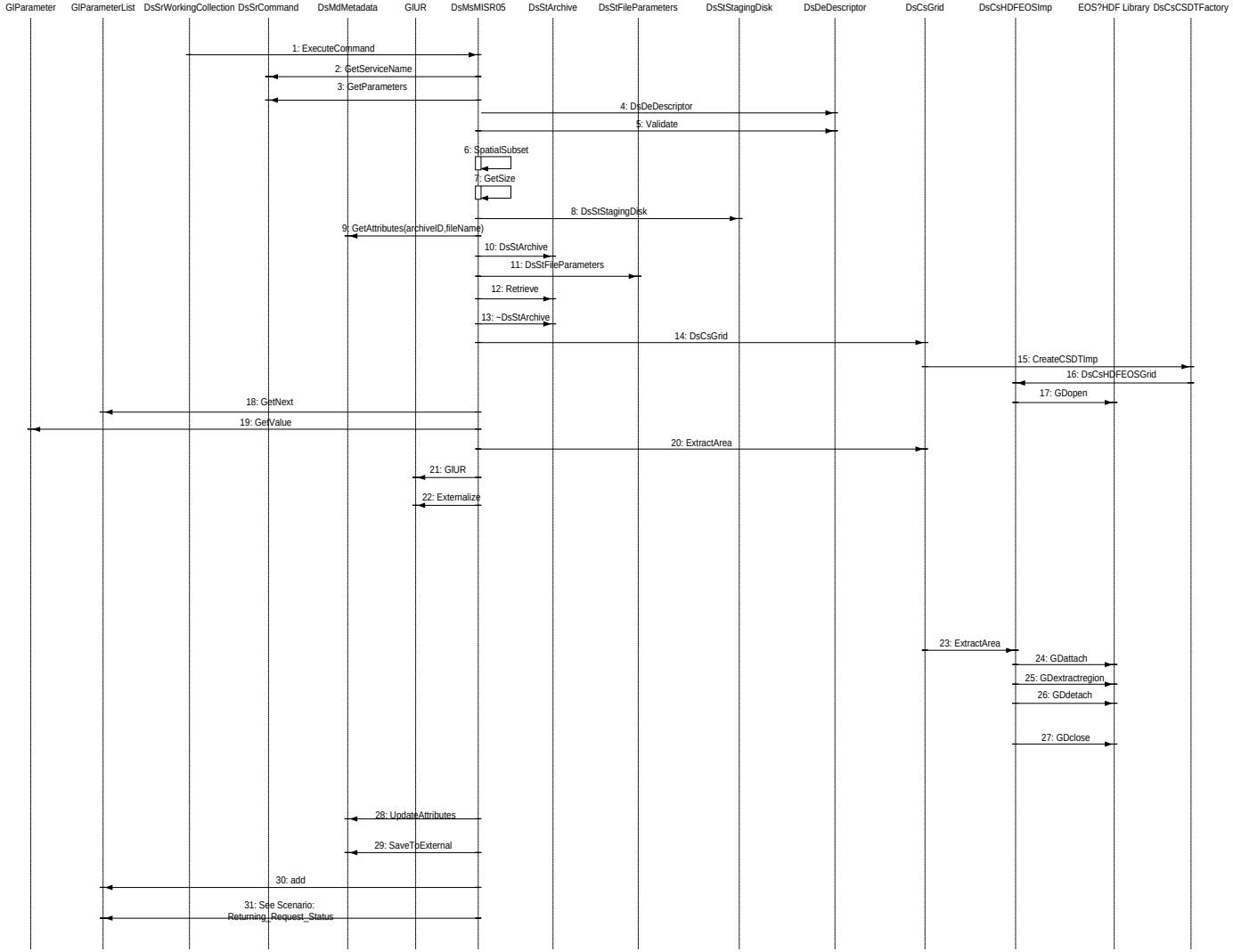


Figure 84. SDSRV Subsetting and ESDT

4.5.26 SDSRV Suspend and Resume 1 Suspend Initial Closing

4.5.26.1 Summary

The purpose of this scenario is to illustrate how the SDSRV supports the suspension and resumption of client sessions. Clients can suspend their session in order to retain the context of their SDSRV activity, while freeing up resources used in support of the session.

All SDSRV clients interact with the SDSRV within a session. Since the SDSRV is a stateful server (that is, it retains context from one request to the next for each of its clients), each session has its own context. A client can have multiple concurrent sessions with the SDSRV. This allows clients to relate to the same server with independent sets of activity. Indeed, there are no cross session capabilities provided within the DSS. The context of a session includes the identification of the client, the current collection of granules that the client is referencing, the service request that have been submitted and the resources utilized in fulfillment of the requests.

The resumption of the session will re-establish the state of the session. Clients are not required to resume a session from the original client hardware platform. They can re-establish their session from any physical location within the ECS domain.

The implementation of a client's session with the SDSRV resides in both the server and the client application. The client application uses the SDSRV client class libraries to establish a session with the SDSRV (by constructing a DsClESDTReferenceCollector object), relating to a collection of ESDT granules (DsClESDTReference objects, collected by the DsClESDTReferenceCollectior) and submitting a series of requests for services (DsClCommands within a DsClRequest). SDSRV software maintains the concurrency between the state of the session within the client application and the server application.

The SDSRV will only retain suspended session for a duration to be set by DAAC operations personnel. If a suspended session has not be resumed, the client is first warned that the session will be deleted in a certain (configurable) time via email, then when the time is exceeded the saved state of the session is removed from the data holdings. At that point the session has been completely removed from the system and is not reconstructable.

For readability, this scenario has been decomposed into the following sub-scenarios :

Client side - Initial request to suspend session

Client side - suspend the session

Server side - suspend the session

Client side - resume the session

Server side - resume the session

Server side - reconstructing an object

Server side - restoring the session

The initial request from the client stimulates the classes in the client and SDSRV to prepare the session for suspension. The request to suspend the session is handled by responding to the client with a session UR and a status indicating that the session has been suspended. This is primarily so that the client can redirect its resources to another session, or close the connection to the SDSRV. However, in both the client and server application, clean up activities are continuing in the back-

ground, ensuring that any currently executing activities are allowed to complete, and that their results are being saved for future restoration of the session.

In this scenario the client application has requested to suspend their currently active session. The request to suspend the session is made by invoking the SetState operation in the DsClESDTReferenceCollector class in the client. The SDSRV handles this request as described in the scenario, “SDSRV Server Request handling”, and determines that it is a request to set the current session to “Suspend”. The session is given a unique ID by the DsClSession UR class on the server side, by which the client can subsequently refer to the saved session. The UR for the suspended session and the status for the request are then written to the server side request class, where it is available to the client. The suspended session information is identified by the application specific portion of the session UR that is given to the client. At this point, the client and server begin their clean up activities, as described in the next few scenarios.

4.5.26.2 Assumptions

- Client has established a session (see Scenario SDSRV Client Connecting to a Dataserver)
- Client has performed a search and has at least one UR in the working collection
- Client has submitted multiple requests (see Scenario SDSRV Client Request Submission)

•The request to suspend a session is given the highest priority in the request queue so that the server can expedite the necessary session suspend activities

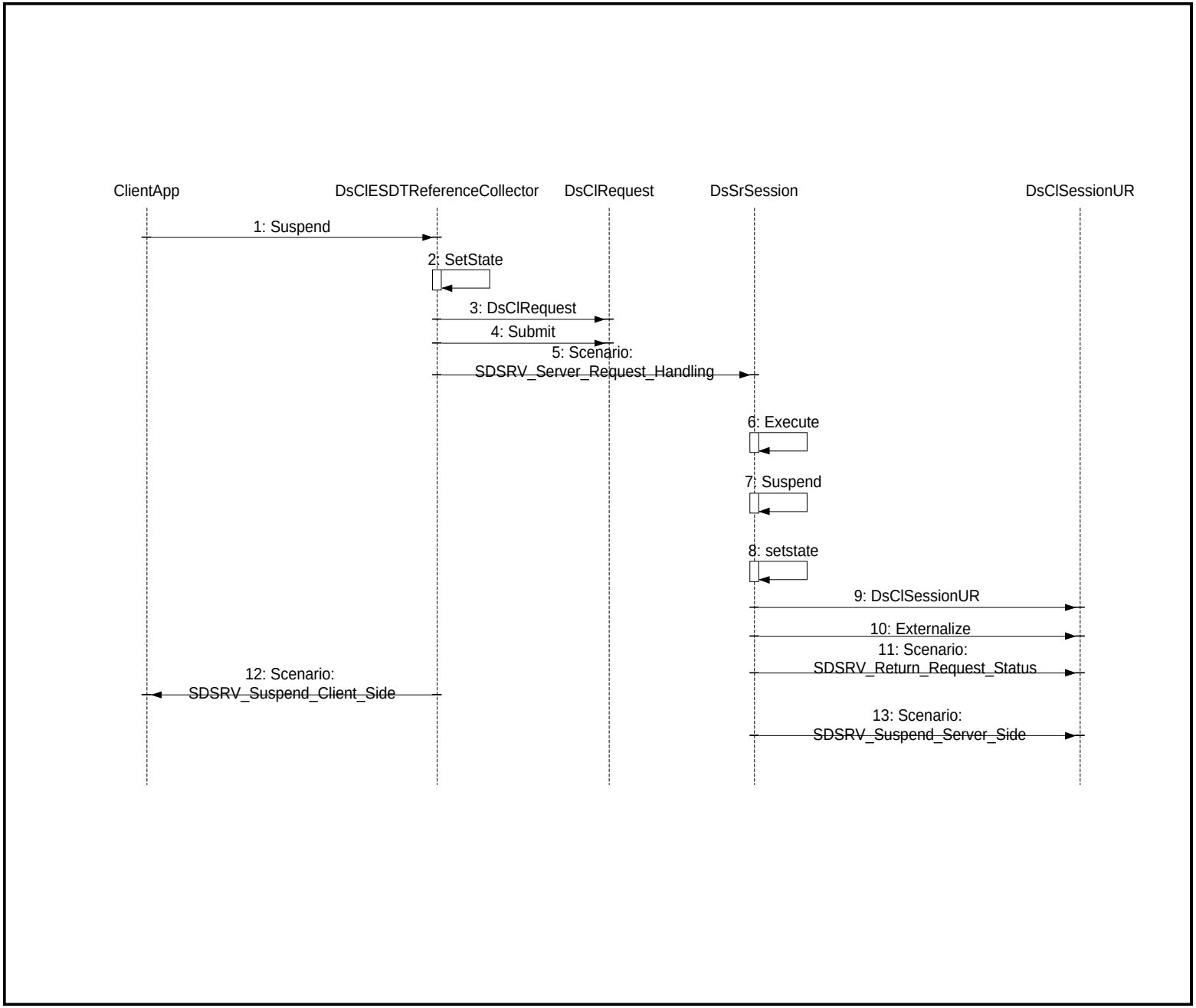


Figure 85 SDSRV Suspend and Resume 1 Suspend Initial Closing

4.5.27 SDSRV Suspend and Resume 2 Suspend Client Side

4.5.27.1 Summary

Within the Client application, all supporting SDSRV client classes are cleaned up. The classes for request submittal (DsClRequest) and client side request handling (DsShSRequestProxy) are deconstructed, and the connection to the SDSRV is closed.

4.5.27.2 Assumptions

•The client will only be able to resume this suspended session after the server has completed its session suspend activities

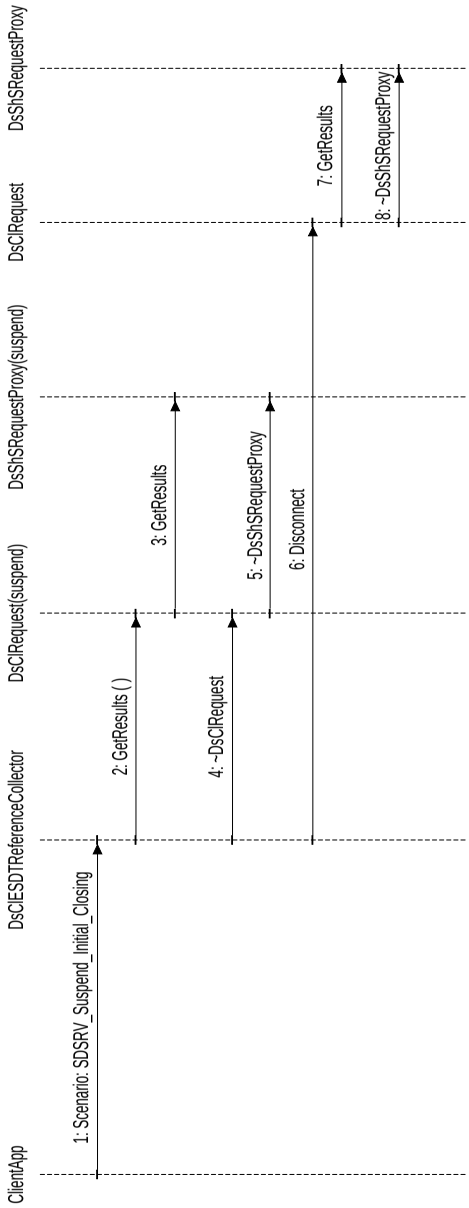


Figure 86: SDSR V Suspend and Resume 2 Suspend Client Side

4.5.28 SDSRV Suspend and Resume 3 Suspend Server Side

4.5.28.1 Summary

Within the SDSRV server application, the current state of the session's request queue and the references to the collection of ESDT granules are captured and saved within a special type of ESDT that contains suspended SDSRV sessions. The session object, DsSrSession instantiates a DsGeSuspendedSession class, that will insert its session information into the archive using the Store operation provided by the DsStArchive object. The DsGeSuspendedSession then inserts its session UR metadata into the catalog using the Insert Granule operation provided by the DsMdCatalog object. At that point, the resources used with the session are deallocated and the objects used within the session deleted. The session object, DsSrSession, deletes the server side request object (DsShRequestReal), the server side working collection (DsSrWorkingCollection), and the classes which assisted in saving the session information (DsMdCatalog, DsStArchive, DsMdMetadata, DsGeSuspendedSession, DsGeESDT).

4.5.28.2 Assumptions

- The client has successfully suspended their session
- The server has completed servicing any active requests connected to the users session

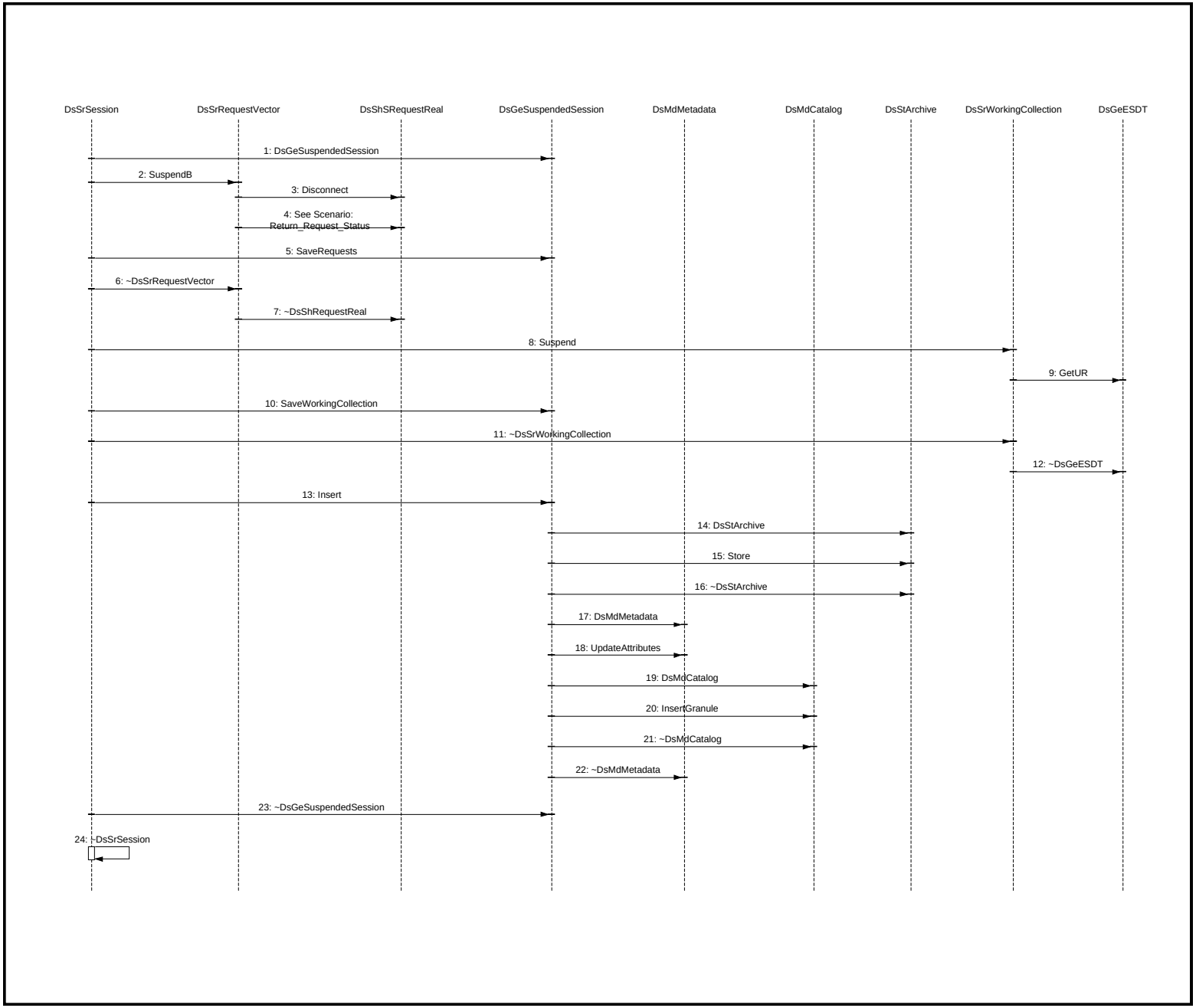


Figure 87 DSR V Suspend and Resume 3 Suspend Server Side

4.5.29 SDSRV Suspend and Resume 4 Resume Session Client Side

4.5.29.1 Summary

In order to resume a session, the client will reconstruct the DsClESDTReferenceCollector with the DsClSessionUR. This identifies the specific session that has been stored within the SDSRV server application. The server will then go through a reconstruction process.

4.5.29.2 Assumptions

•Client has inititated Session resumption

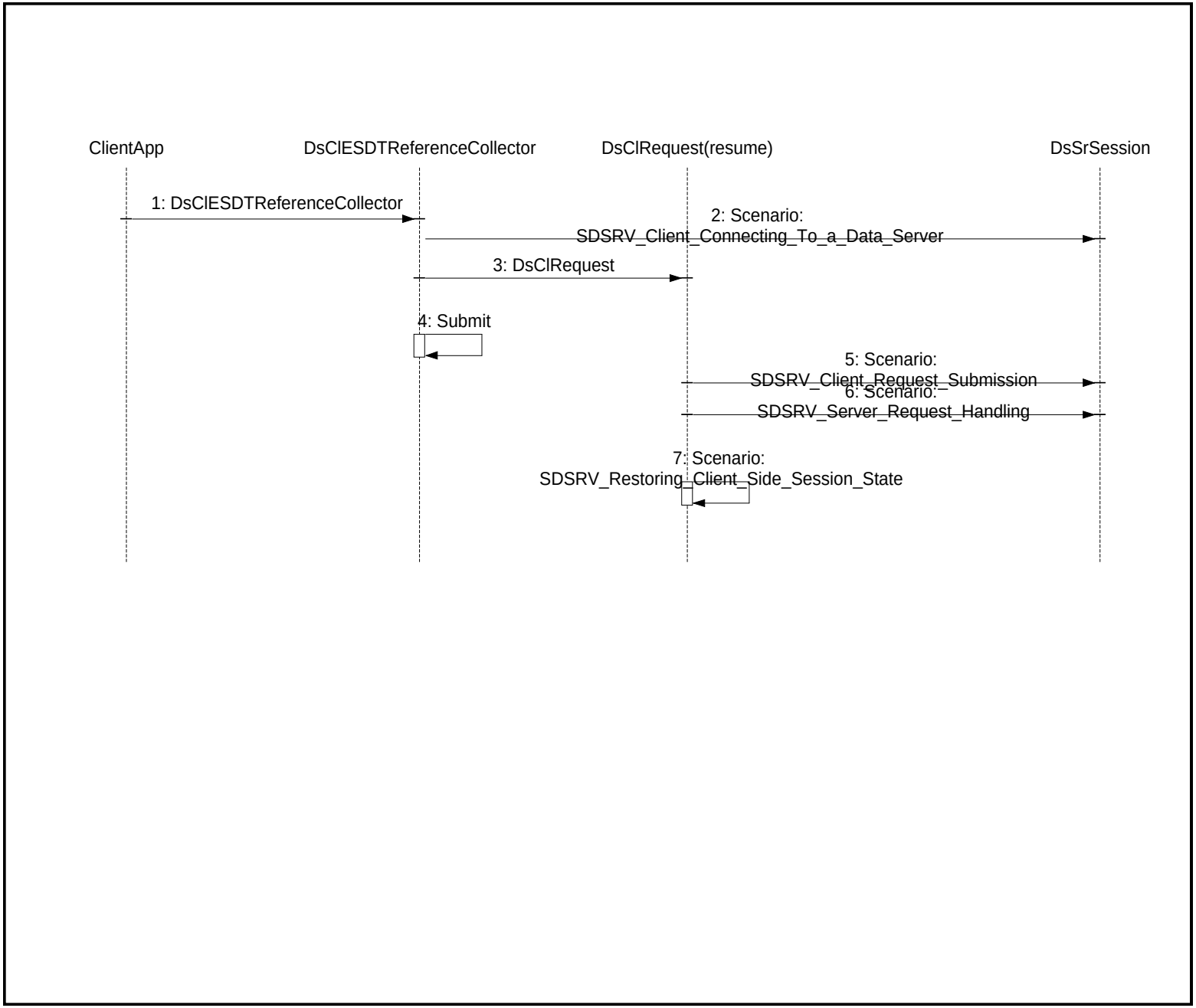


Figure 88 SDSRV Suspend and Resume 4 Resume Session Client Side

4.5.30 SDSRV Suspend and Resume 5 Resuming Session Server Side

4.5.30.1 Summary

The server re-establishes the working collection for the session, and populates it with the saved UR from the user's previously suspended session. Subsequent steps in this scenario are to reconstruct the DsGeSuspendedSession are in "Reconstructing a Ge Suspended Object".

4.5.30.2 Assumptions

•The client has the saved UR from a previously suspended session

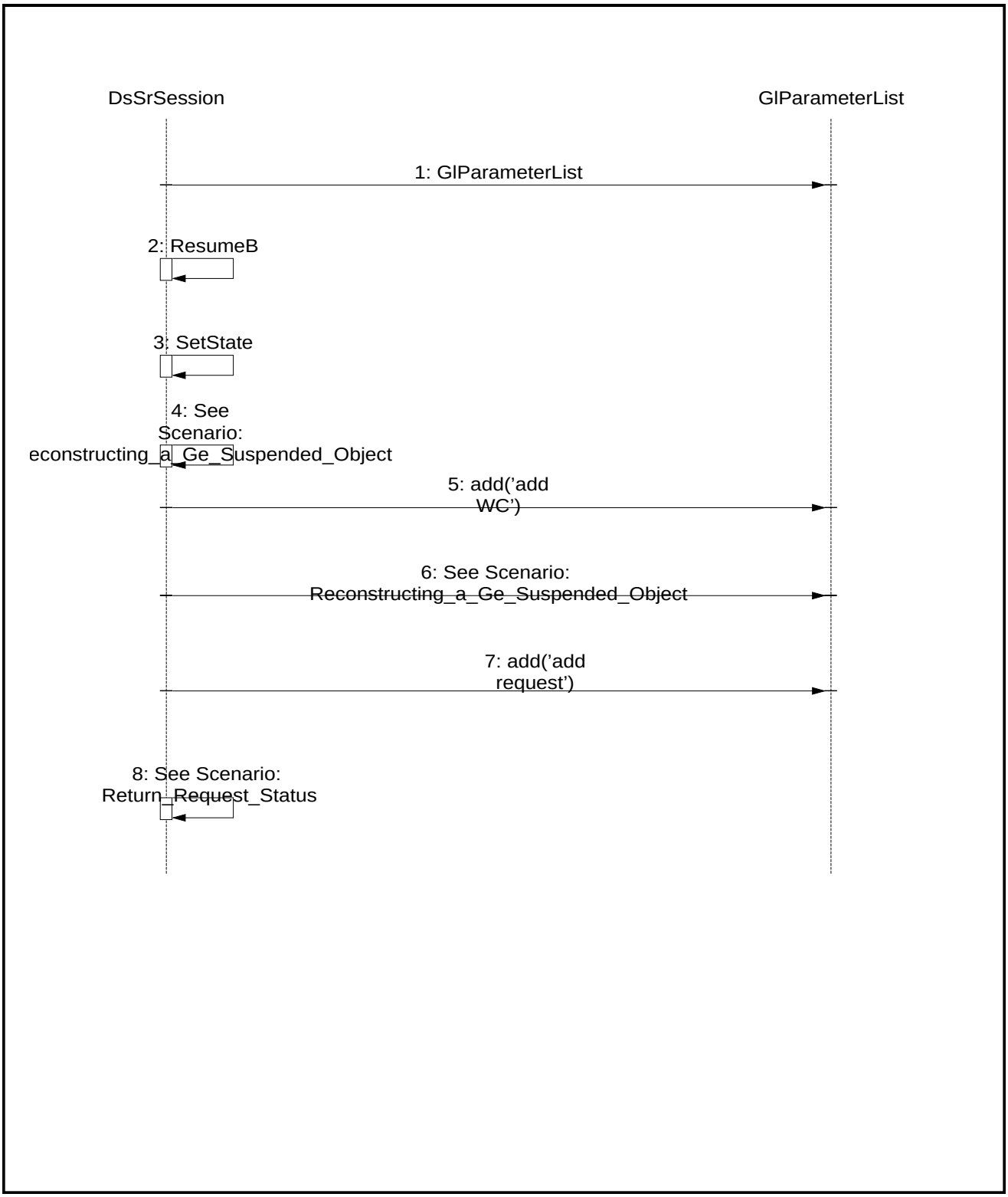


Figure 89 SDSR V Suspend and Resume 5 Resuming Session Server Side

4.5.31 SDSRV Suspend and Resume 6 Reconstructing A Ge Suspended Object

4.5.31.1 Summary

The request queues are rebuilt and the SDSRV client classes then reconstruct the state of the session within the client application. When both sides are complete, the server will then pass all saved results to the client (as requests might have continued to completion after the client-side was suspended).

4.5.31.2 Assumptions

•The session requested for resumption is identified by a valid UR, which correctly identifies the saved session information in the archive

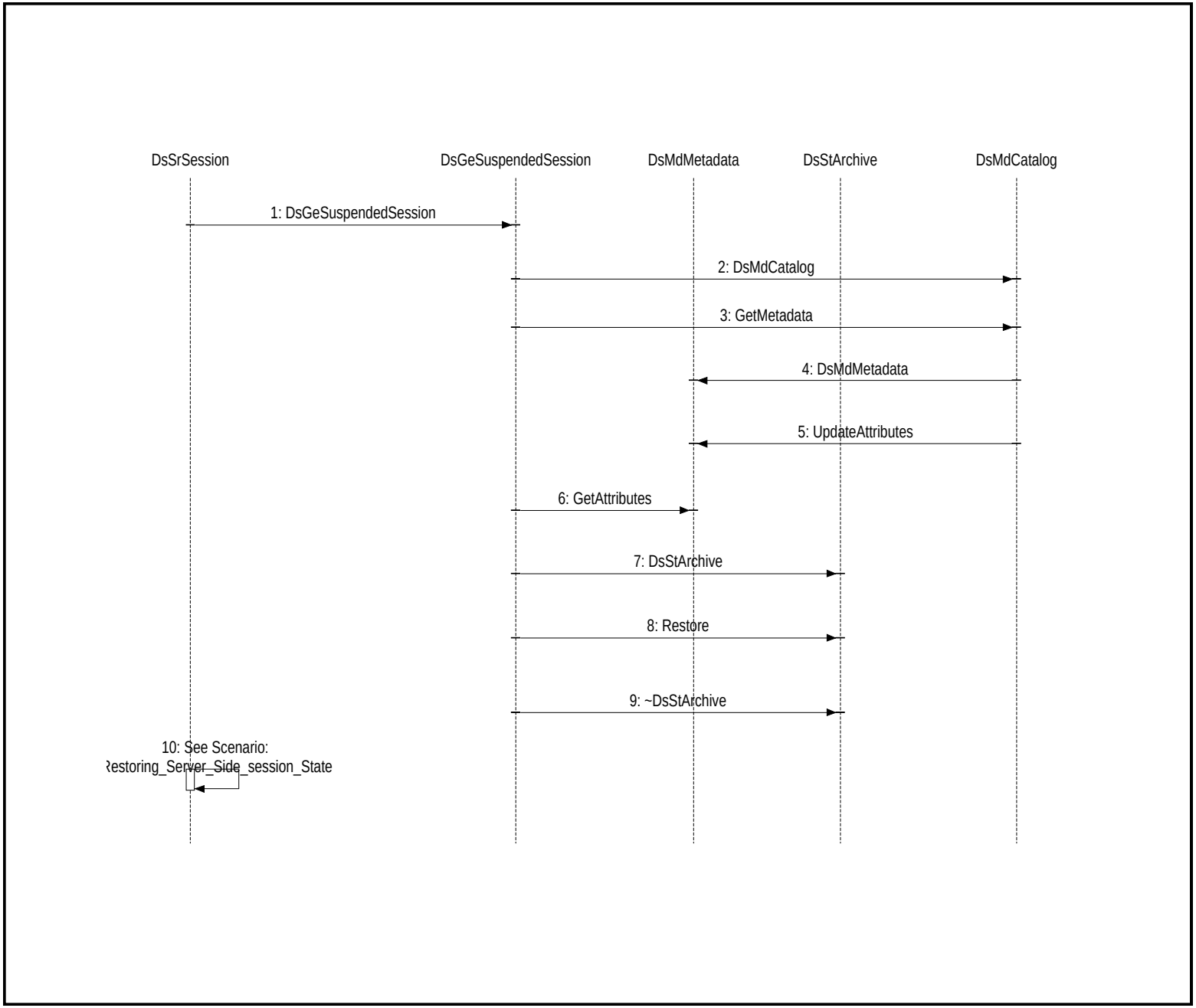


Figure 90: DSR V Suspend and Resume 6 Reconstructing A Ge Suspended Object

4.5.32 SDSRV Suspend and Resume 7 Restoring Server Side Session

4.5.32.1 Summary

Once the results are synchronized, the server picks up servicing the request queue and control is returned back to the client application for further client activity.

4.5.32.2 Assumptions

•The server has successfully identified reconstructed the DsSrWorking collection using the saved session information

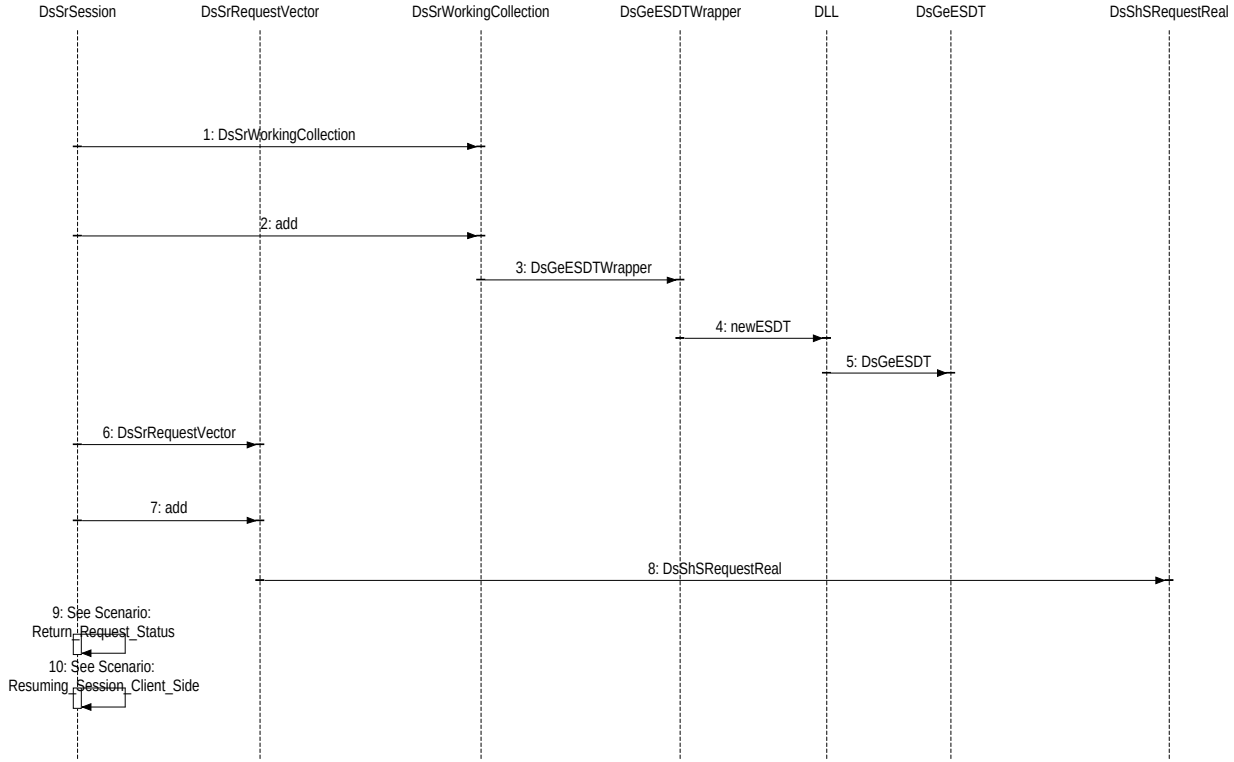


Figure 91 DSR V Suspend and Resume 7 Restoring Server Side Session

4.5.33 SDSRV Suspend and Resume 8 Restoring Client Side Session State

4.5.33.1 Summary

The client software constructs a DsClESDTReferenceCollector with the GlUR of the previously suspended session. The DsClESDTReferenceCollector establishes a connection to the server (see Scenario SDSRV Client Connecting to a Dataserver) and the server reconstructs the suspended session (see SDSRV Server Resuming a Session). Once the connection has been established, the DsClESDTReferenceCollector invokes the GetState method of the DsSrSession. The DsClESDTReferenceCollector then performs the UpdateState operation to reconstruct the matching state on the client-side. This involves creating the DsClESDTReferenceVector, which in turn creates a DsClTypeInfo, and the various DsClESDTReferences. The DsClESDTReferenceVector queries the DsSrSession for a list of all the requests. It then performs the BuildRequestVector operation, which constructs a DsClRequestVector and populates it with newly created DsClRequests. The DsClESDTRequestVector then signals the DsSrSession to "resume" (i.e., begin to accept requests from the client), sets its own state to "active," and returns from the constructor.

4.5.33.2 Assumptions

•Client successfully reconstructs the DsClESDTReferenceCollector with the GlUR of the previously suspended session and completes its connection to the server

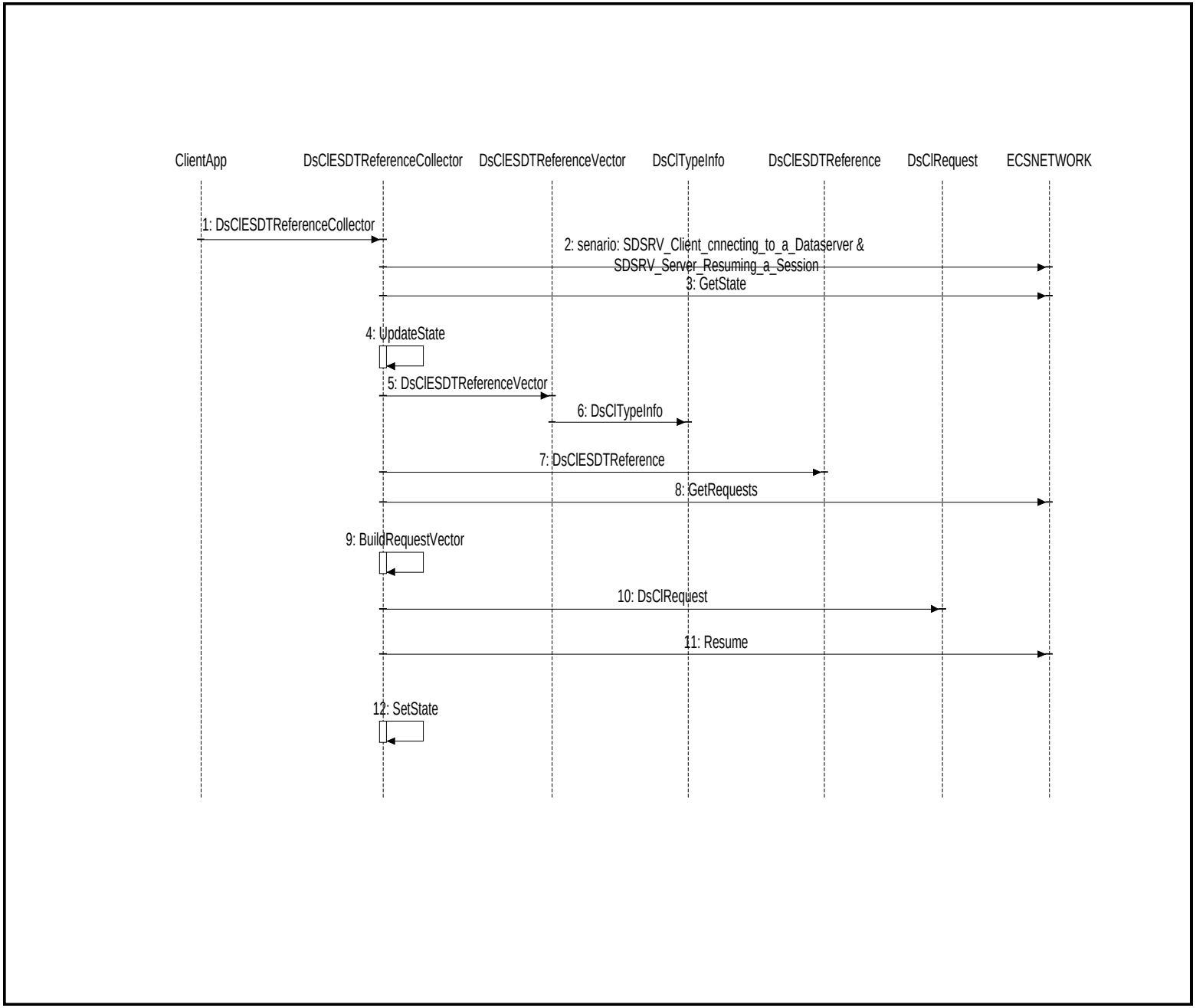


Figure 92 SDSRV Suspend and Resume 8 Restoring Client Side Session State

4.5.34 SDSRV Update Server Configuration

4.5.34.1 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 from the SDSRV GUI. The operator then selects an appropriate action widget (e.g. button) that triggers a callback, in this case the SetQueueSize method of DsAdRequestInterface object.

4.5.34.2 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.

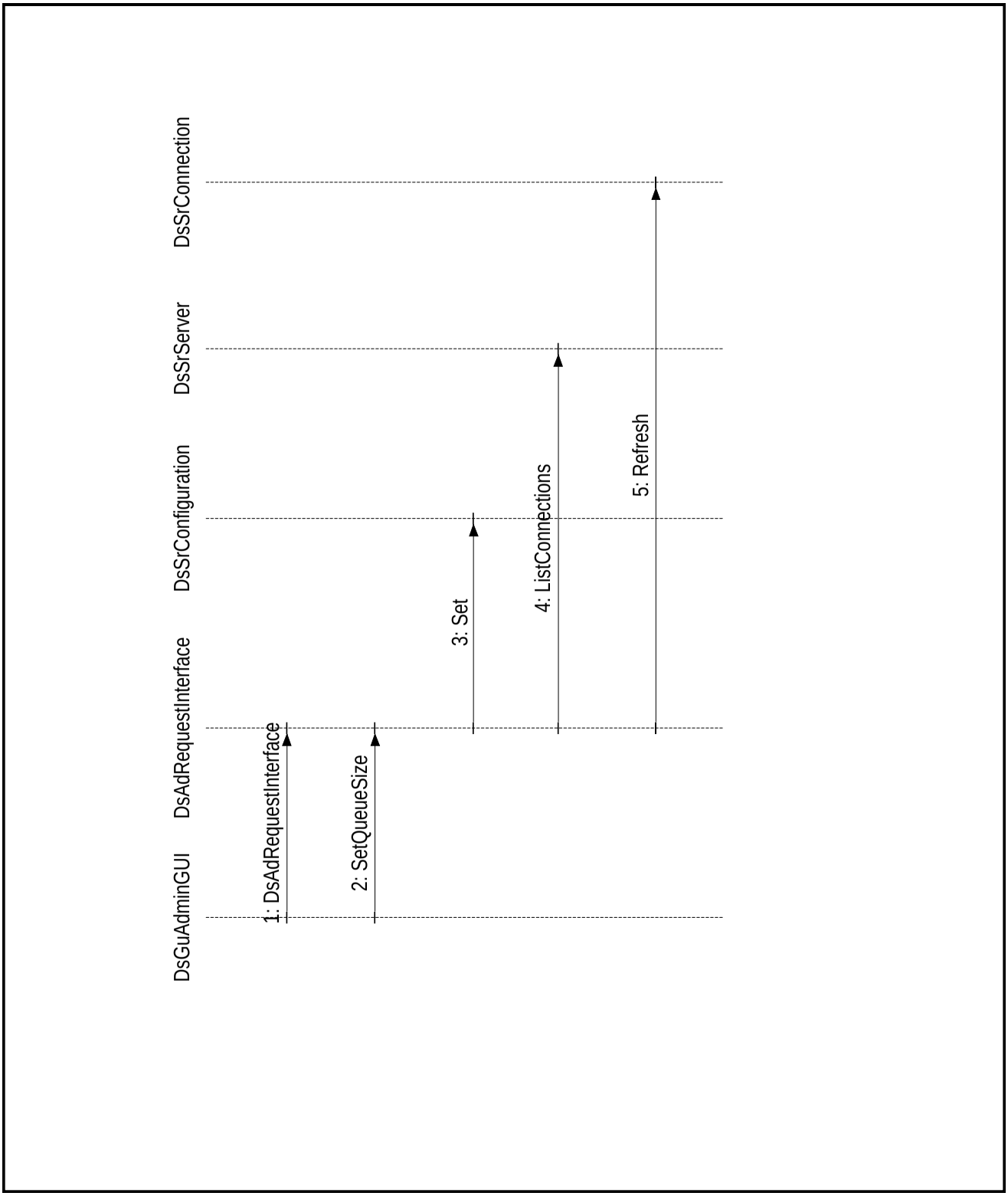


Figure 93 SDSR V Update Server Configuration

4.5.35 SDSRV Validating Metadata

4.5.35.1 Summary

The purpose of this scenario is to show how validation of metadata is accomplished within the Science Data Server. Metadata validation is performed SDSRV whenever a new granule is inserted or its metadata is updated. This scenario is primarily used by the Ingest and Processing Subsystems. This scenario assumes that the validation is being done in the context of an insertion of a new granule. Validation of metadata includes both a completeness check (that all required metadata fields are specified) and a value check (that all values are valid).

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 DsDeESDT-Descriptors 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 (Product Specific Metadata). These valids are a refinement of the DsDeCoreValids. This process is nearly identical to the CoreValid-Vector 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.

4.5.35.2 Assumptions

- The client of this service provided a P=V metadata stream.

•All metadata fields are valid.

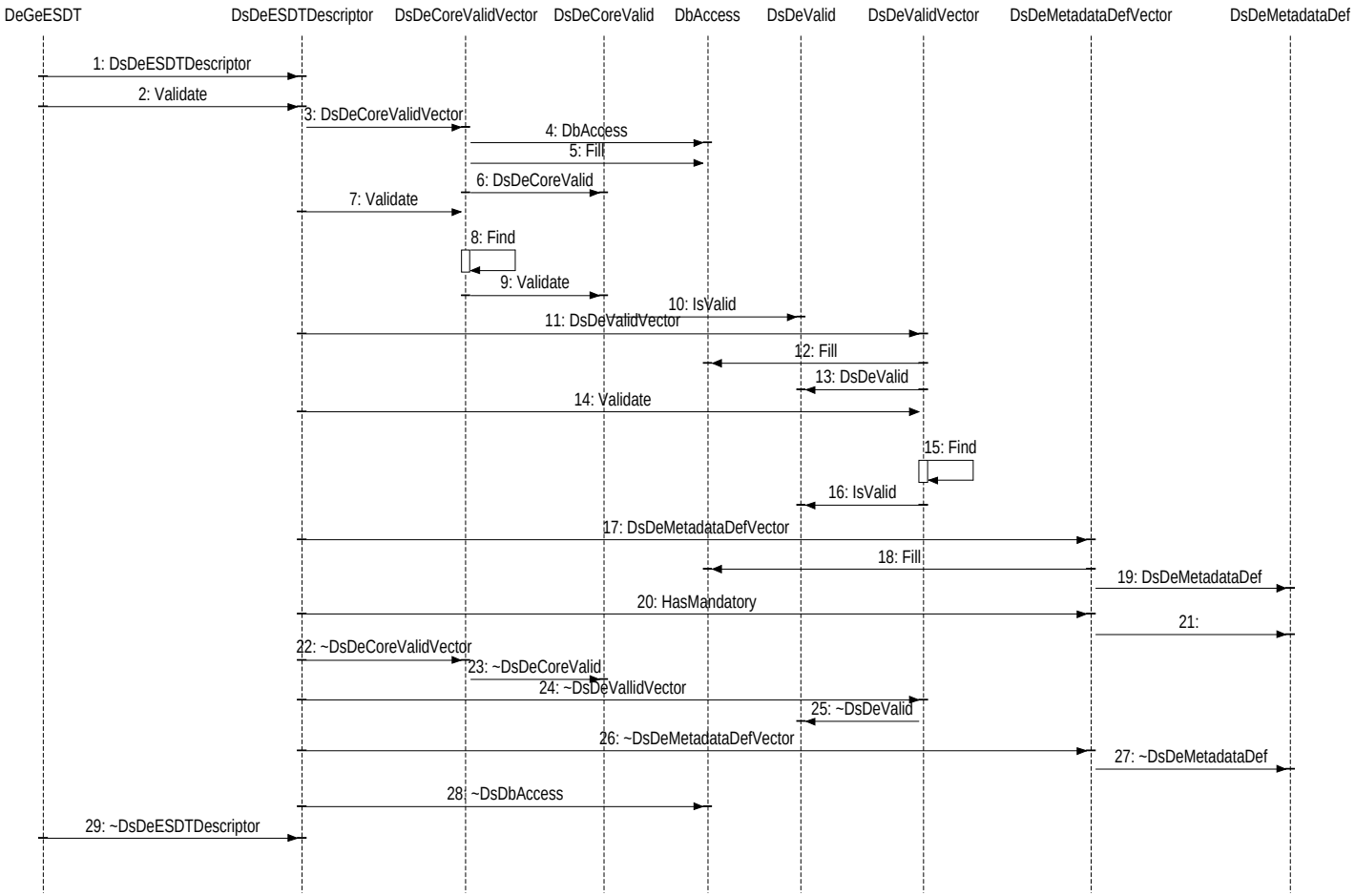


Figure 94 SDSR V Validating Metadata

4.6 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 (1 of 2)

CSC	Abbr.	Description	Implementation
ACRIM	Ac	The classes supporting the ACRIM Instrument Data Products.	DEV
Administration/Operation	Ad	The classes required to support Administrator and Operations functionality.	DEV
ASTER	As	The classes supporting the ASTER Instrument Data Products	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 start-up of the SDSRV.	DEV
COLOR	Co	The classes supporting the COLOR Instrument Data Products.	DEV
Metadata	Md	The classes and database implementations supporting the storage and access of metadata.	DEV/WRAPPER
CSDT	Cs	The CSDT classes.	WRAPPER
DAO	Da	The classes supporting data products from the DAO.	DEV
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
DFA/MR	Dm	The classes supporting the DFA and MR Instrument Data Products	DEV
ERS	Er	The classes supporting the data products from instruments on ERS.	DEV
ETM	Et	The classes supporting the ETM Instrument Data Products.	DEV
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

Table 4.5-1. SDSRV's Components (2 of 2)

CSC	Abbr.	Description	Implementation
Ground Validation RADAR	Gv	The classes supporting the Ground Validation RADAR products.	DEV
JERS	Je	The classes supporting data products from the instruments on JERS.	DEV
MODIS	Mo	The classes supporting the MODIS Instrument Data Products.	DEV
MOPITT	Mp	The classes supporting the MOPITT Instrument Data Products.	DEV
MISR	Ms	The classes supporting the MISR Instrument Data Products.	DEV
NMC	Nm	The classes supporting data products from the NMC.	DEV
Non-Product ESDTs	Np	The classes supporting science data that was not produced within ECS.	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
RADARSAT	Ra	The classes supporting data products from the instruments on Radarsat.	DEV
SAGE	Sa	The classes supporting the SAGE Instrument Data Products.	DEV
Shared	Sh	The classes providing support across all of the SDSRV CSCs	DEV
Server	Sr	The classes used to provide the framework for the infrastructure of the SDSRV.	DEV/OTS
SeaWinds	Sw	The classes supporting the SeaWinds Instrument Data Products.	DEV
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. ACRIM

Purpose and Description

The classes supporting the ACRIM Instrument Data Products.

Classes

DsAcACRIM

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.2. Administration/Operation

Purpose and Description

The classes required to support SDSRVAdmininstrator and Operations functionality.

Classes

DsAdBaseInterface
DsAdConfigurationInterface
DsAdDatatypeInterface
DsAdDescriptor
DsAdRequestInterface
DsAdSubscriptionInterface

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.3. ASTER

Purpose and Description

The classes supporting the ASTER Instrument Data Products.

Classes

DsAsAsterB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.4. CERES

Purpose and Description

The classes supporting the CERES Instrument Data Products.

Classes

DsCeCer00
DsCeCer02
DsCeCERES

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.5. Client

Purpose and Description

The Public classes offered to SDSRV clients. This includes distributed classes.

Classes

DsClCommand
DsClDescriptor
DsClESDTReferenceCollectorVector
DsClESDTReferenceCollector
DsClESDTReferenceVector
DsClESDTReference
DsClGenConnector
DsClGenRequestInt
DsClGenSuspendor
DsClInsertCommand
DsClQuery
DsClRequest
DsClTypeInfo

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.5. 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

4.5.1.6. COLOR

Purpose and Description

The classes supporting the COLOR Instrument Data Products.

Classes

DsCoColorB
DsCoCombination

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.7. CSDT

Purpose and Description

The CSDT classes.

Classes

DsCs24BitImage
DsCs8BitImage
DsCsBinaryPacketImp
DsCsBinaryPackets
DsCsCSDTFactory
DsCsCSDTImp
DsCsCSDT
DsCsGrid
DsCsHDFEOSImp
DsCsImage
DsCsLookupTable
DsCsPoint
DsCsRaw
DsCsSwath

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.8 DAO

Purpose and Description

The classes supporting data products from the DAO.

Classes

DsDaDAO

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.9.DB Wrappers

Purpose and Description

The COTS database wrapper classes.

Classes

DsDbAttributeTableXref
DsDbAttributeToTableVector
DsDbEngine
DsDbEventStore
DsDbGranuleToDbVector
DsDbInterface
DsDbProductDbXref

Candidate Products

Illustra

ECS White Paper References

Not Applicable

4.5.1.10.Descriptors

Purpose and Description

The classes used to define and describe ESDTs and their supporting functionality.

Classes

DsDeAttNode
DsDeAttributeContent
DsDeAttributeType
DsDeConvert
DsDeCustomizerVector
DsDeCustomizer
DsDeDDCustomizerVector
DsDeDDCustomizer
DsDeDescriptorBuilder
DsDeDescriptorImp
DsDeDescriptorSet
DsDeDescriptor
DsDeDictionary
DsDeEventCustomizerVector
DsDeEventCustomizer
DsDeEventTableInfo
DsDeEventTable
DsDeExpressionRule
DsDeGroupNode
DsDeMatchRule
DsDeMCFtoODL

DsDeMetadataCustomizerVector
DsDeMetadataCustomizer
DsDeNode
DsDeRangeRule
DsDeRealExpressionRule (ParameterizedClass)
DsDeRealMatchRule (ParameterizedClass)
DsDeRealRangeRule (ParameterizedClass)
DsDeRealRuleFactory
DsDeRealRule
DsDeRuleFactory
DsDeRule
DsDeServiceCustomizerVector
DsDeServiceCustomizer
DsDeServiceVector
DsDeStaticMetadataVector
DsDeStaticMetadata
DsDeStructCustomizerVector
DsDeStructCustomizer
DsDeSymbolicExpressionRule
DsDeSymbolicMatchRule
DsDeSymbolicRangeRule
DsDeSymbolicRuleFactory
DsDeSymbolicRule

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.11.DFA/MR

Purpose and Description

The classes supporting the DFA and MR Instrument Data Products.

Classes

DsDmDFAMR

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.12.ERS

Purpose and Description

The classes supporting the data products from instruments on ERS.

Classes

DsErERSB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.13.ETM

Purpose and Description

The classes supporting the ETM Instrument Data Products.

Classes

DsEtETMB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.14.General ESDT

Purpose and Description

Those classes which provide the framework around which specific ESDTs are built.

Classes

DsGeBrowseProduct
DsGeDynamicLibrary
DsGeESDTConfiguration
DsGeESDTDynamicLibrary
DsGeESDTEventTable
DsGeESDTServiceProvider
DsGeESDTWrapper.
DsGeESDTWrapper
DsGeESDT
DsGeOID
DsGeScienceData
DsGeSummaryProduct
DsGeSuspendedSession
DsGeTypeID

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.15.Global

Purpose and Description

Those classes that are intended to be globally available for re-use.

Classes

GlBinaryP
GlCircleP
GlDateP
GlDoubleP
GlGPolygonP
GlLogHelper
GlLog
GlLongP
GlParameterBase
GlParameterList
GlParameter
GlParmType
GlPfxstreambuf
GlPfxstream
GlPointP
GlPointSetP
GlPointSet
GlPoint
GlPolygonP
GlPolygonSetP
GlPolygonSet
GlPolygon
GlRectangleP
GlSpatialP
GlStringP
GlTimeP

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.16.GUI

Purpose and Description

The classes and other software that is supplied to support whatever GUI requirements that the SD-SRV has.

Classes

DsGuAdmin

DsGuConfigurationMgmt
DsGuDatatypeMgmt
DsGuRequestMgmt

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.17.Ground Validation RADAR

Purpose and Description

The classes supporting the Ground Validation RADAR Data Products.

Classes

DsGvRadar

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.18.JERS

Purpose and Description

The classes supporting data products from the instruments on JERS.

Classes

DsJeJERSB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.19.Metadata

Purpose and Description

The classes and database implementations supporting the storage and access of metadata.

Classes

DsMdAttributeList
DsMdCatalog
DsMdID
DsMdJoinTable
DsMdMetadata
DsMdODL

DsMdPSTableAliasList
DsMdReachablePair

Candidate Products

Illustra

ECS White Paper References

Not Applicable

4.5.1.20.MODIS

Purpose and Description

The classes supporting the MODIS Instrument Data Products.

Classes

DsMoMODISB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.21.MOPITT

Purpose and Description

The classes supporting the MOPITT Instrument Data Products.

Classes

DsMpMOPPITB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.22.MISR

Purpose and Description

The classes supporting the MISR Instrument Data Products.

Classes

DsMsMISRB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.23.NMC

Purpose and Description

The classes supporting the data products from the National Meteorological Center (NMC).

Classes

DsNmNMC

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.24.Non-Science ESDTs

Purpose and Description

The classes used to support the ESDTs that do not contain Earth Science Data.

Classes

DsNsHistoricalDataB
DsNsHistoricalData
DsNsMPRB
DsNsNonECSDataProduct
DsNsProdPlans
DsNsProductionHistory
DsNsQAStatistics
DsNsScienceSoftwareArchivePackage

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.25.PR

Purpose and Description

The classes supporting the PR Data Products.

Classes

DsPrRadar

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.26.RADARSAT

Purpose and Description

The classes supporting data products from instruments on RADARSAT.

Classes

DsRaRadarsatB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.27.SAGE

Purpose and Description

The classes supporting the SAGE Instrument Data Products.

Classes

DsSaSageB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.28Shared

Purpose and Description

The classes providing functionality across the SDSRV CSCs, but primarily used by the Client (Cl) and Server (Sr) CSCs.

Classes

DsShActionBase
DsShConnectionReal
DsShConnection_X_Y
DsShDCEConnectionConcrete
DsShDCEDescriptorConcrete
DsShDCEId
DsShDCEInterface
DsShDCERef
DsShDCESRequestConcrete
DsShDCEUpdaterConcrete
DsShDescriptorProxyDCE
DsShDescriptorProxy
DsShDescriptorRealDCE
DsShDescriptorReal

DsShDOError
DsShDOId
DsShDOInterface
DsShDONotFound
DsShDORef
DsShError
DsShGenCommandImp
DsShGenRequestImp
DsShGlobal
DsShMutexLock
DsShMutexValVector
DsShMutex
DsShQuitProxy
DsShQuitReal
DsShSciCommandImp
DsShSciRequestImp
DsShSRequestProxy
DsShSRequestReal
DsShSRequest_X_Y
DsShTypeID
DsShUpdateable
DsShUpdaterProxy
DsShUpdaterReal
DsShUpdater_X_Y_ABS
DsShUpdater_X_Y

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.29.Server

Purpose and Description

The classes used to provide the framework for the infrastructure of the SDSRV.

Classes

DsSrArchiveCostB
DsSrClient
DsSrCommand
DsSrConnectionMaker
DsSrCostB
DsSrCostPolicyB
DsSrCostTableB
DsSrCPUUtilizationB
DsSrDescriptorMaker

DsSrDiskUtilizationB
DsSrEnvVariable
DsSrEnv
DsSrFixedPersonnelCostB
DsSrGenCatalogPool
DsSrGenCatalogWrapper
DsSrGenConnection
DsSrGenQueuedConnection
DsSrGenRequestInt
DsSrGenRequestVector
DsSrGenRequest
DsSrGenServer
DsSrIOUtililizationB
DsSrMediaUtilizationB
DsSrRequestExample
DsSrRequestVector
DsSrRequest
DsSrResourceB
DsSrServer
DsSrSession
DsSrWorkingCollection

Candidate Products

RogueWave

ECS White Paper References

Not Applicable

4.5.1.30.SeaWinds

Purpose and Description

The classes supporting the SeaWinds Instrument Data Products.

Classes

DsSwSeaWindsB

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.31.TMI

Purpose and Description

The classes supporting the TMI Data Products.

Classes

DsTmTMI

Candidate Products

Not Applicable

ECS White Paper References

Not Applicable

4.5.1.32.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 two separate, cooperating UNIX processes. Those processes are the ScienceDataServer and the SDSRVAdmin. In addition, the COTS DBMS Illustra Server supports storage of and access to SDSRV metadata and documents. Illustra provides a single daemon process as well as a server process per connection to Illustra. The ScienceDataServer, SDSRVAdmin, and Illustra processes, along with those executables that include the SDSRV Client CSC library, 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 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 B implementation of the ScienceDataServer Process will comprise implementations for the Release A data types as well as Release B data types. These implementations are the ACRIM, ASTER, CERES, COLOR, ERS, ETM, JERS, LIS, MISR, MODIS, MOPITT, PR, RADARSAT, SAGE, DFA/MR, SeaWinds, 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 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 General Application of SDSRV Client CSC

The applications that utilize ScienceDataServer Process capabilities are generically referred to as "SDSRV Client Applications", or "Clients" within the SDSRV context. There are a number of ECS SDSRV Clients Applications as well as, potentially, non-ECS SDSRV Client Applications. Examples of ECS Clients are the Ingest, Planning, Processing and Data Management 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.

4.7 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. At DSS system startup, all of the required SDSRV 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 ScienceDataServer process acts as an "object factory". As such the process instantiates objects in process threads (pthreads) when a request is serviced.

The Illustra processes are also started by MSS when DSS is started. Additional server processes are started as additional connections are made to Illustra by the ScienceDataServer process. The Illustra ORDBMS provides 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 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 (1 of 2)

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 Client Fatal Error	This condition occurs whenever a Data Base client API library call is made that returns what is considered to be a "fatal" (i.e., connection is terminated) error from the COTS Data Base. An example of this would be an internal software error from the COTS Data Base.
DB Server Fatal Error	This condition occurs when, for example, the Data Base, is full, but includes other scenarios where the connection to the Data Base server is closed, the current transaction is aborted, and processing of the current Data Base command stops.
DB Connection Unavailable	This condition occurs when an initial connection cannot be established to the 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.
DB Client Warning Error	This type of error is raised when non-connection terminating errors occur in various Data Base API calls. A typical example occurs when an incorrect argument is passed to an API routine, such as a bad connection description or a NULL value where a pointer is required.
DB Server Warning Error	This type of error occurs when processing of the current transaction is aborted and processing of the current Data Base command is halted, but the connection to the server is unaffected.

Table 4.6-1. SDSRV Error Categories (2 of 2)

Error Category	Actions to Be Taken
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.

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 will also be used in support of the suspension and resumption of a client's session. 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
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

5 DDSRV - Document Data Server CSCI

5.1 CSCI Overview

The Document Data Server (DDSRV) reflects many of the design principles used by the SDSVR in its management of Earth Science Data Types. The DDSRV 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 are services" 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 DDSRV maximizes the reuse of common functionality with the SDSRV. Common base classes are reused 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, and have been implemented with a specialization of the CSDT class.

Services are statically bound to document objects. A stateless communication paradigm is used, thereby reducing design complexity. Object association multiplicity has been minimized within the design, reflecting the design drivers behind the DDSRV.

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 DDSRV and SDSVR, a single common data model will be internally managed in order to support the eventual navigation between science and document objects, thus 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 client interface to the DDSRV is provided through HTTP and a class library. Search and acquire requests are formulated as WAIS queries and HTTP Get commands respectively and issued by the Client using HTTP. A HTTP demon process monitors a TCP/IP port for incoming requests and hands them off to the DDSRV using a CGI (Common Gateway Interface) 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 DDSRV. The document is inserted into the document repository; the metadata is inserted into the DBMS through the DBMS Wrapper layer, and the document is then indexed for free text queries using COTS indexing.

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.

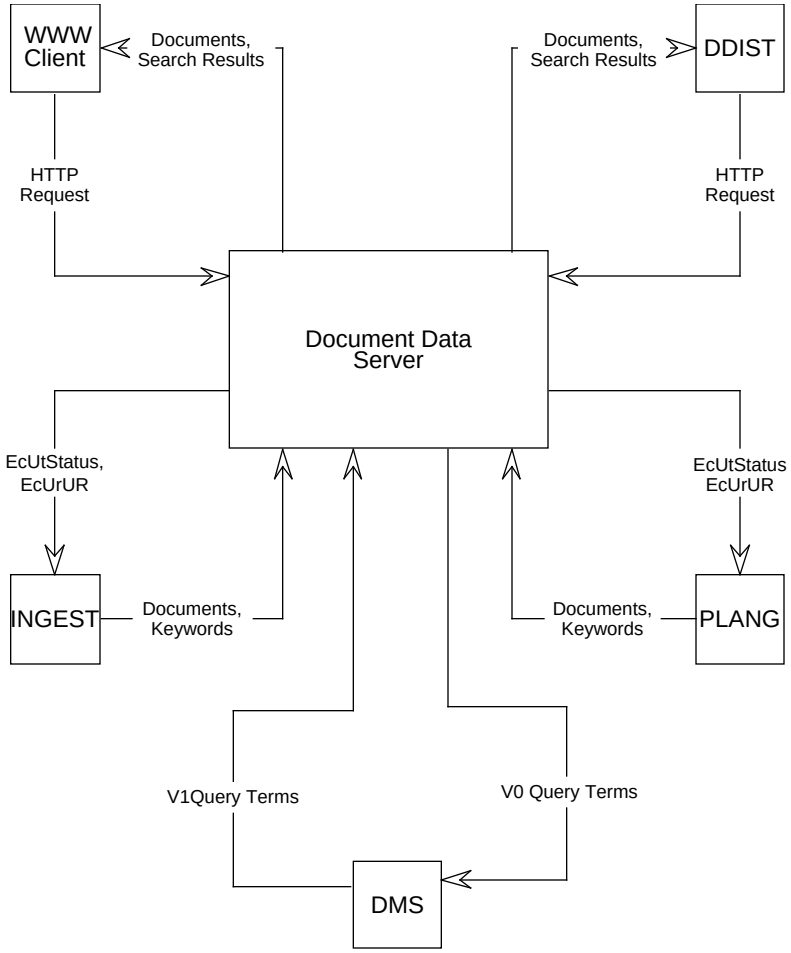


Figure 5.2-1 DDSRV Client Interfaces

Table 5.2-1. DDSRV Client Interfaces

Interface	Interface Input	Interface Output	Description
InsertCommand	DsCtInsertCommand:: InsertCommand (DsCtInsertCommandList)	GlStatus	This interface is used by DDSRV clients to insert documents and their metadata into the DDSRV
WAIS Search Command	DsCtSearchCommand:: ExecuteSearch (DsCtSearchCommandList)	GlStatus	This interface is used by DDSRV clients to request WAIS formatted searches on DDSRV document metadata
Keyword Search Command	DsCtSearchCommand:: ExecuteKeywordSearch (DsCtSearchCommandList)	GlStatus	This interface is used by DDSRV clients to request keywords searches on DDSRV document metadata
Free Text Search Command	DsCtSearchCommand :: ExecuteFreeTextSearch (DsCtSearchCommandList)	GlStatus	This interface is used by DDSRV clients to request free text searches against a document index
Acquire document	DsCtAcquireCommand :: AcquireCommand (DsCtAcquireCommandList)	GlStatus	This interface us used by non-HTTP DDSRV clients to request documents

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 which support the document types of 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, Binary, Word, WordPerfect, and Interleaf). 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 in-

herited 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).

5.3.1 cd Class Category

5.3.1.1 Overview

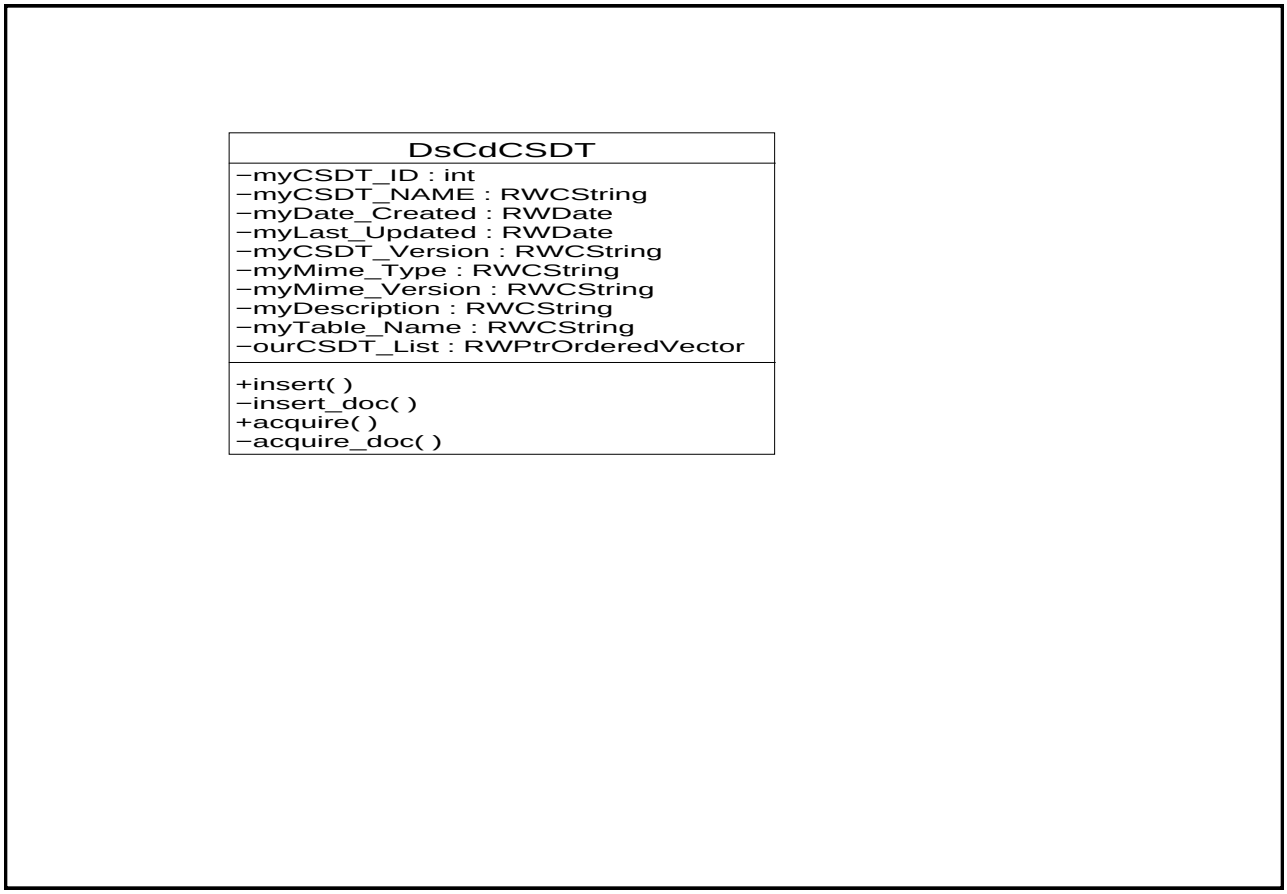


Figure 95 Main

5.3.1.2 cd Classes

5.3.1.2.1 DsCdCSDT Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myCSDT_ID: int

myCSDT_NAME: RWCString

myCSDT_Version: RWCString

myDate_Created: RWDate

myDescription: RWCString

myLast_Updated: RWDate

myMime_Type: RWCString

myMime_Version: RWCString

myTable_Name: RWCString

ourCSDT_List: RWPtrOrderedVector
```

Constructors and Destructor:

Operations:

- acquire_doc

```
acquire_doc();
```

- acquire

```
EcUtStatus acquire(dbname, DocumentID);
```

- acquire

```
EcUtStatus acquire(dbName, DocumentID, RWvostream);
```

- insert_doc

```
insert_doc(DbName, iostream);
```

- insert

```
EcUtStatus insert(dbName, fileName);
```

- insert

```
EcUtStatus insert(dbName, RWvistream);
```

5.3.2 ct Class Category

5.3.2.1 Overview

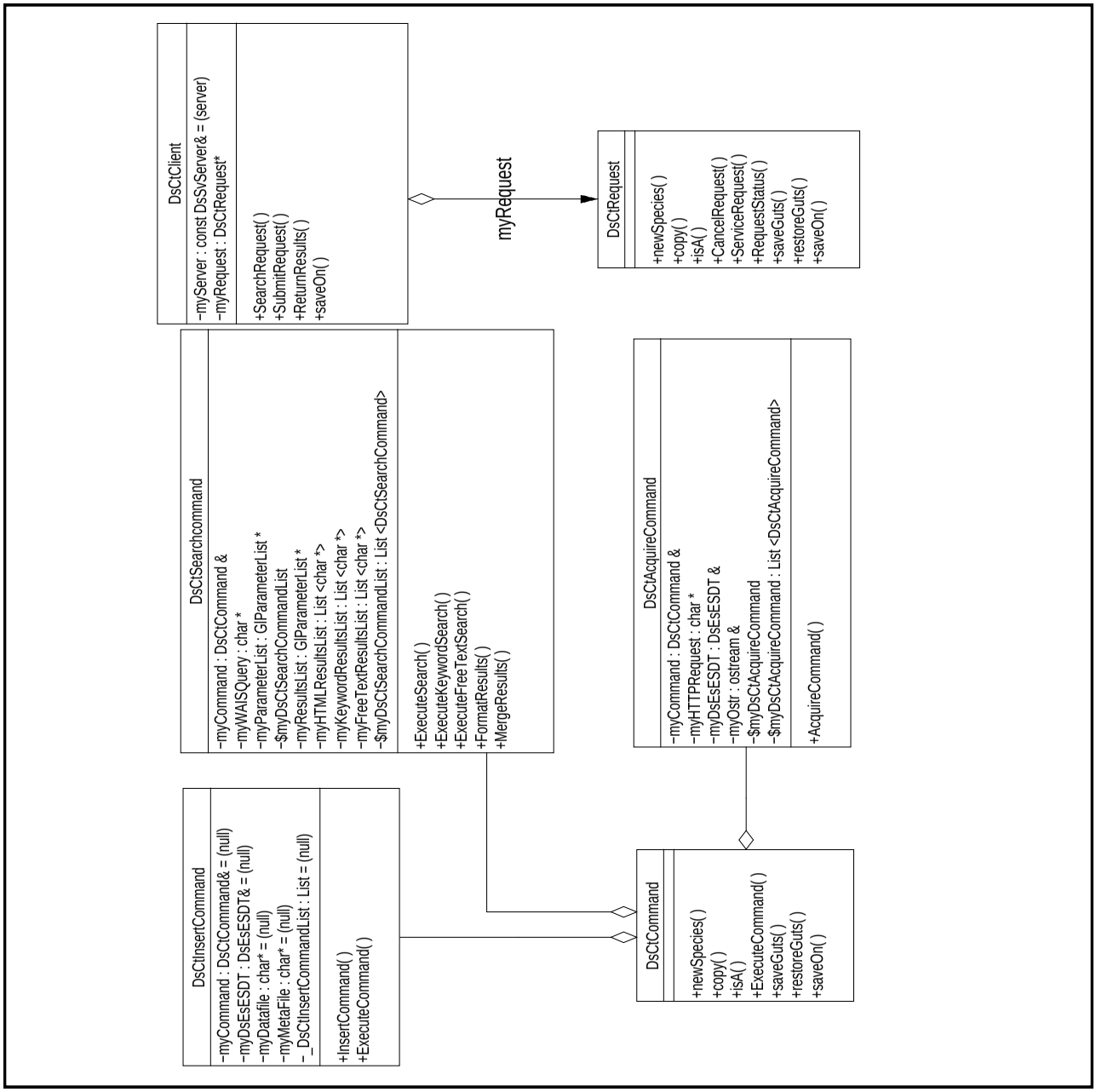


Figure 96 Main

5.3.2.2 ct Classes

5.3.2.2.1 DsCtAcquireCommand Class

Overview:

Distributed Object: Yes This object represents the document retrieval commands received for the document data.

Export Control: Public

Inheritance Relationships:

Attributes:

`$myDsCtAcquireCommand: List <DsCtAcquireCommand>`

List of all active acquire commands. Used mainly for fault recovery and memory management.

`myCommand: DsCtCommand &`

Reference to the associated command.

`myDsCtAcquireCommand:`

`myDsEsESDT: DsEsESDT &`

Reference to the document ESDT associated with the acquire command. The CSDT related to this ESDT is used to implement the extract operation.

`myHTTPRequest: char *`

HTTP Get command string. Used to identify the location of the document to return.

`myOstr: ostream &`

Output stream to write document data.

Constructors and Destructor:

`DsCtAcquireCommand(DsCtCommand &);`

This object represents the constructor for the commands received for acquiring the Document Data.

`~DsCtAcquireCommand();`

Object destructor - invoke ESDT destructor.

Operations:

- AcquireCommand

  `GlStatus AcquireCommand();`

  This is the actual acquire command received for the Document data.

5.3.2.2.2 DsCtClient Class

Overview:

Distributed Object: Yes

Export Control: Public

Inheritance Relationships:

Inherits from DsDoClient

Attributes:

`myRequest: DsCtRequest*`

Reference to server

`myServer: const DsSvServer&`

Constructors and Destructor:

`DsCtClient(const RWCString& clientName, const DsSvServer& server);`

Name of client

`DsCtClient(const RWCString& clientName);`

Default constructor

`DsCtClient();`

Reference to DsCtClient

`~DsCtClient(void );`

to output stream BEGIN_PROLOG

NAME ~DsCtClient

DESCRIPTION Default destructor

RETURNS void

GLOBALS none

FILES none
END_PROLOG

Operations:

- operator ==

  ```
  int operator ==(const DsCtClient& client);
  ```

  BEGIN_PROLOG
  NAME operator==
  DESCRIPTION Equality operator.
  RETURNS RWBoolean
  GLOBALS none
  FILES none
  END_PROLOG

- ReturnResults

  ```
  GlStatus ReturnResults(GlParameterList* results) const;
  ```

  Submit request to server BEGIN_PROLOG
  NAME ReturnResults
  DESCRIPTION Return pointer to results.
  PHASE2 - restore results from file PHASE3 - retrieve results over distributed connection.
  RETURNS GlStatus
  GLOBALS none
  FILES none
  END_PROLOG

- saveOn

  ```
  void saveOn(ostream& os) const;
  ```

  Points to results parameters BEGIN_PROLOG
  NAME saveOn(ostream &os) const;
  DESCRIPTION Global stream operator for DsCtClient objects.
  RETURNS void
  GLOBALS none
  FILES none
  END_PROLOG

- SearchRequest

```
GlStatus SearchRequest(const RWCString& name, const
GlParameterList& params);
```

Reference to DsSvServer BEGIN_PROLOG

NAME SearchRequest

DESCRIPTION Construct a search request

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- SubmitRequest

```
GlStatus SubmitRequest();
```

Request parameters BEGIN_PROLOG

NAME SubmitRequest

DESCRIPTION Submit request to server.

PHASE2 - flatten request to file PHASE3 - pass request over distributed connection.

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

5.3.2.2.3 DsCtCommand Class

Overview:

Distributed Object: Yes

Export Control: Public

Inheritance Relationships:

Inherits from DsDoCommand

Attributes:

Constructors and Destructor:

```
DsCtCommand();
```

```
DsCtCommand(const RWCString& svc, const GlParameterList& pl,
enum DsDoCommand::_typetag typ);
```

Default construtor

```
~DsCtCommand();
```

Execute this command BEGIN_PROLOG

NAME ~DsCtCommand

DESCRIPTION Destructor

RETURNS void

GLOBALS none

FILES none

END_PROLOG

Operations:

- copy

```
virtual RWCollectable* copy() const;
```

- ExecuteCommand

```
GlStatus ExecuteCommand();
```

Command type BEGIN_PROLOG

NAME ExecuteCommand();

DESCRIPTION Execute command.

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

to file BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

from stream BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

Destructor BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

to stream BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

  `void saveOn(ostream& os) const;`

  from file BEGIN_PROLOG

  NAME saveOn(ostream &os) const;

  DESCRIPTION Global stream operator for DsShCommandBase objects.

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

5.3.2.2.4 DsCtInsertCommand Class

Overview:

Class: DsCtInsertCommand Distributed Object: Yes Description: This object represents the insert commands received for the document data.

SubSystemID: DDSRV

Export Control: Public

Inheritance Relationships:

Attributes:

`myCommand: DsCtCommand&`

description of attribute myCommand

`myDatafile: char*`

description of attribute myDatafile

`myDsEsESDT: DsEsESDT&`

description of attribute myDsEsESDT

`myMetaFile: char*`

description of attribute myMetaFile

`_DsCtInsertCommandList: List`

description of attribute _DsCtInsertCommandList

Constructors and Destructor:

```
DsCtInsertCommand(DsCtCommand& );
```

DsCtInsertCommand

description of operation

Operations:

- ExecuteCommand

```
ExecuteCommand();
```

- InsertCommand

```
GlStatus InsertCommand(void );
```

InsertCommand

description of operation ----------

DsCtInsertCommand::InsertCommand

5.3.2.2.5 DsCtRequest Class

Overview:

Distributed Object: Yes

Export Control: Public

Inheritance Relationships:

Inherits from DsDoRequest

Attributes:

Constructors and Destructor:

```
DsCtRequest();
```

```
DsCtRequest(const RWCString& requestName, const DsSvServer&
server, const DsCtClient& client, enum DsDoRequest::_typetag
typ);
```

Default constructor

```
~DsCtRequest(void );
```

PHASE3 BEGIN_PROLOG

NAME ~DsCtRequest

DESCRIPTION Destructor

RETURNS void

GLOBALS none

FILES none

END_PROLOG

Operations:

- CancelRequest

```
GlStatus CancelRequest(void );
```

Internal or External

- copy

```
virtual RWCollectable* copy() const;
```

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- RequestStatus

```
GlStatus RequestStatus(void );
```

Service this request BEGIN_PROLOG

NAME <FUNCTION_NAME>

DESCRIPTION PHASE3

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWvistream& vis);
```

to file BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
void restoreGuts(RWFile& f);
```

from stream BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWFile& f) const;
```

to stream BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
void saveGuts(RWvostream& vos) const;
```

PHASE3 BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

from stream BEGIN_PROLOG

NAME saveOn(ostream &os) const;

DESCRIPTION Global stream operator for DsCtRequest objects.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- ServiceRequest

```
GlStatus ServiceRequest(void );
```

Default destructor BEGIN_PROLOG

NAME ServiceRequest

DESCRIPTION Service the command in request

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

5.3.2.2.6 DsCtSearchCommand Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

```
myCommand: const DsCtCommand&
```

```
myECSQueryResults: GlParameterList*
```

Reference to command object

```
myFreeTextResults: GlParameterList*
```

Returned from ECS keyword search

```
myResults: GlParameterList*
```

Returned from freetext search

Constructors and Destructor:

```
DsCtSearchCommand(const DsCtCommand& command);
```

```
~DsCtSearchCommand(void );
```

Reference to command object BEGIN_PROLOG
NAME ~DsCtSearchCommand
DESCRIPTION Destructor
RETURNS void
GLOBALS none
FILES none
END_PROLOG

Operations:

- ExecuteFreeTextSearch

```
GlStatus ExecuteFreeTextSearch(const GlParameter* freetext);
```

Parameters for keyword search BEGIN_PROLOG
NAME ExecuteFreeTextSearch
DESCRIPTION Perform freetext search
RETURNS GlStatus
GLOBALS none
FILES none
END_PROLOG

- ExecuteKeywordSearch

```
GlStatus ExecuteKeywordSearch(const GlParameter* keywords);
```

Combined results BEGIN_PROLOG

NAME ExecuteKeywordSearch

DESCRIPTION Perform keyword search

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- ExecuteSearch

```
GlStatus ExecuteSearch(void );
```

Destructor BEGIN_PROLOG

NAME ExecuteSearch

DESCRIPTION Execute search command

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- MergeResults

```
GlStatus MergeResults(void );
```

Parameters for freetext search BEGIN_PROLOG

NAME MergeResults

DESCRIPTION none

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- ReturnResults

```
GlStatus ReturnResults(void );
```

Merge the results BEGIN_PROLOG

NAME ReturnResults

DESCRIPTION Return results to calling function

RETURNS GlStatus

GLOBALS none

FILES none
END_PROLOG

5.3.3 do Class Category

5.3.3.1 Overview

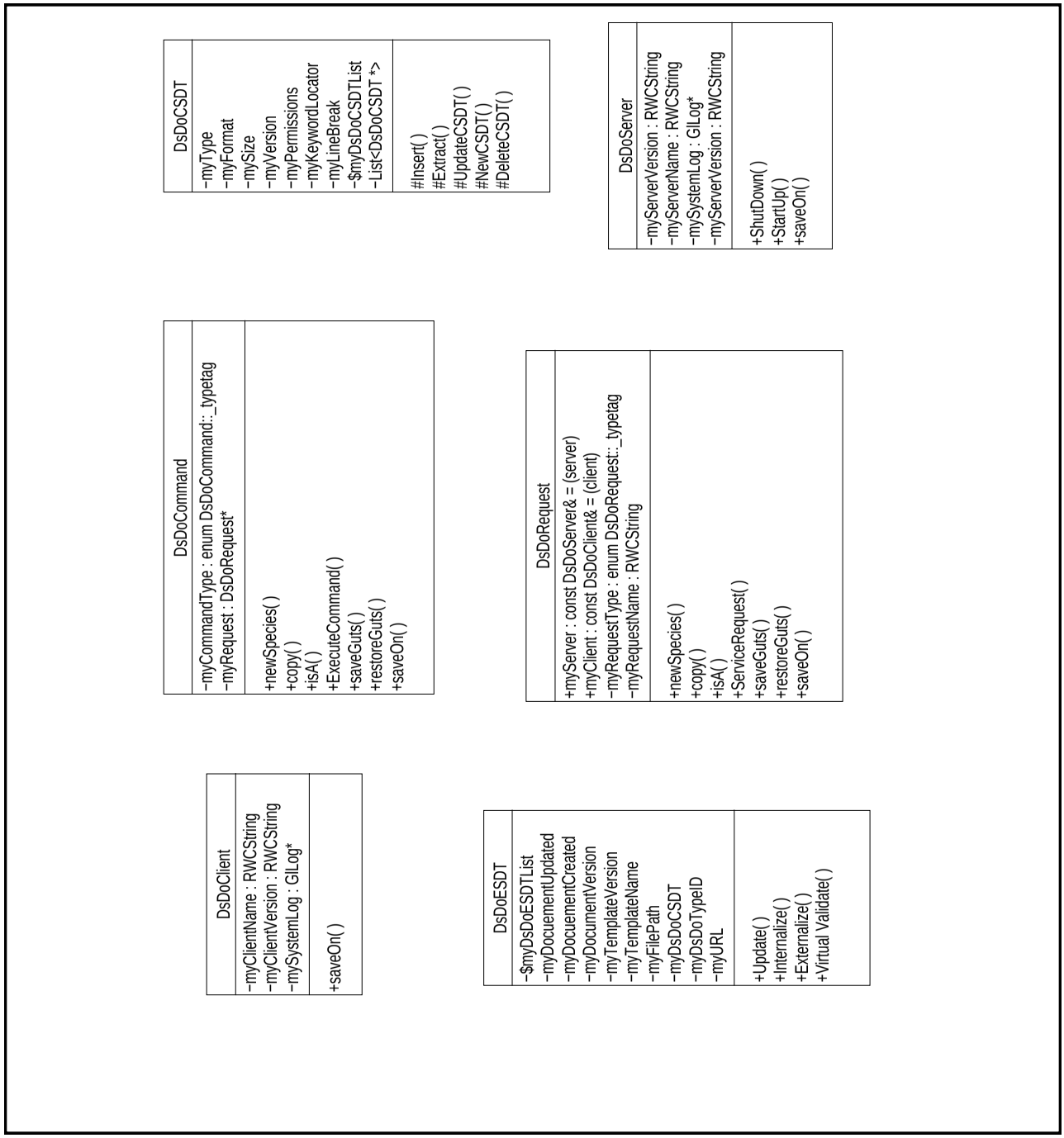


Figure 97 Main

5.3.3.2 do Classes

5.3.3.2.1 DsDoClient Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myClientName: RWCString`

`myClientVersion: RWCString`

Name of this client

`mySystemLog: GlLog*`

Version of this client

Constructors and Destructor:

`DsDoClient();`

BEGIN_PROLOG

NAME DsDoClient

DESCRIPTION Default constructor. Uses default ClientName DsDoClientName

RETURNS void

GLOBALS none

FILES none

END_PROLOG

`DsDoClient(const RWCString& clientName);`

Default constructor BEGIN_PROLOG

NAME DsDoClient(const RWCString &clientName)

DESCRIPTION Default constructor. Sets myClientName to

RETURNS void

GLOBALS none

FILES none

END_PROLOG

Operations:

- saveOn

```
void saveOn(ostream& os) const;
```

BEGIN_PROLOG

NAME saveOn(ostream &os) const;

DESCRIPTION Global stream operator for DsShCommandBase objects.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

5.3.3.2.2 DsDoCommand Class

Overview:

Distributeed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myCommandType: enum DsDoCommand::_typetag
```

```
myRequest: DsDoRequest*
```

Type of command

Constructors and Destructor:

```
DsDoCommand();
```

```
DsDoCommand(const RWCString& svc, const GlParameterList& pl,
enum DsDoCommand::_typetag cat);
```

Default constructor

Operations:

- copy

```
virtual RWCollectable* copy() const;
```

- ExecuteCommand

```
virtual GlStatus ExecuteCommand();
```

Returns command type BEGIN_PROLOG
NAME ExecuteCommand();
DESCRIPTION Submit command to server.
Note: This function should be overridden by the derived class. It should not be called directly.
RETURNS GlStatus
GLOBALS none
FILES none
END_PROLOG

- GetType

```
enum DsDoCommand::_typetag GetType() const;
```

Command type BEGIN_PROLOG
NAME GetType
DESCRIPTION get the current type for this command
RETURNS the category
GLOBALS none
FILES none
END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

to file BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

from virtual stream BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

Submit command to server BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

to virtual stream BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS void

GLOBALS none
FILES none
END_PROLOG

- saveOn

  `void saveOn(ostream& os) const;`
  from file BEGIN_PROLOG
  NAME saveOn(ostream &os) const;
  DESCRIPTION Global stream operator for DsDoCommand objects.
  RETURNS void
  GLOBALS none
  FILES none
  END_PROLOG

- SetType

  `void SetType(enum DsDoCommand::_typetag cat);`
  Type of command BEGIN_PROLOG
  NAME SetType
  DESCRIPTION set the type for the command
  RETURNS void
  GLOBALS none
  FILES none
  END_PROLOG

5.3.3.2.3 DsDoCSDT Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`$myDsDoCSDTList`:

```
List<DsDoCSDT *>:

myFormat:

myKeywordLocator:

myLineBreak:

myPermissions:

mySize:

myType:

myVersion:
```

Constructors and Destructor:

Operations:

- DeleteCSDT

```
DeleteCSDT();
```

- Extract

```
Extract();
```

- Insert

```
Insert();
```

- NewCSDT

```
NewCSDT();
```

- UpdateCSDT

```
UpdateCSDT();
```

5.3.3.2.4 DsDoESDT Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
$myDsDoESDTList:

myDocuementCreated:

myDocuementUpdated:

myDocumentVersion:

myDsDoCSDT:

myDsDoTypeID:

myFilePath:

myTemplateName:
```

```
myTemplateVersion:
```

```
myURL:
```

Constructors and Destructor:

```
DsDoESDT(DsDoTypeID *, DsDoCSDT *);
```

```
~DsDoESDT();
```

Operations:

- Externalize

```
Externalize();
```

- Internalize

```
Internalize();
```

- Update

```
Update();
```

- Virtual Validate

```
Virtual Validate();
```

5.3.3.2.5 DsDoRequest Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myClient: const DsDoClient&
```

Reference to server

```
myRequestName: RWCString
```

```
myRequestType: enum DsDoRequest::_typetag
```

Name of this request

```
myServer: const DsDoServer&
```

to ostream

Constructors and Destructor:

```
DsDoRequest();
```

```
DsDoRequest(const RWCString& requestName, const DsDoServer&
server, const DsDoClient& client, enum DsDoRequest::_typetag
typ);
```

Default constructor

```
 ~DsDoRequest(void );
```

Internal or External BEGIN_PROLOG

NAME ~DsDoRequest

DESCRIPTION Default destructor

RETURNS void

GLOBALS none

FILES none

END_PROLOG

Operations:

- copy

```
virtual RWCollectable* copy() const;
```

- GetType

```
enum DsDoRequest::_typetag GetType() const;
```

Internal or External BEGIN_PROLOG

NAME GetType

DESCRIPTION get the current type for this command

RETURNS the type

GLOBALS none

FILES none

END_PROLOG

- isA

```
virtual unsigned short isA() const;
```

- newSpecies

```
virtual RWCollectable* newSpecies() const;
```

- restoreGuts

```
virtual void restoreGuts(RWFile& f);
```

from stream BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a RWFile

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- restoreGuts

```
virtual void restoreGuts(RWvistream& vis);
```

to file BEGIN_PROLOG

NAME restoreGuts

DESCRIPTION a required RWCollectable function that streams an object from a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWvostream& vos) const;
```

Service this request BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a virtual RW stream

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveGuts

```
virtual void saveGuts(RWFile& f) const;
```

to stream BEGIN_PROLOG

NAME saveGuts

DESCRIPTION a required RWCollectable function that streams an object to a RWFile

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- saveOn

```
void saveOn(ostream& os) const;
```

from file BEGIN_PROLOG

NAME saveOn(ostream &os) const;

DESCRIPTION Global stream operator for DsShCommandBase objects.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- ServiceRequest

```
GlStatus ServiceRequest(void );
```

Returns request type

- SetType

```
void SetType(enum DsDoRequest::_typetag typ);
```

Default destructor BEGIN_PROLOG

NAME SetType
DESCRIPTION set the type for the command
RETURNS void
GLOBALS none
FILES none
END_PROLOG

5.3.3.2.6 DsDoServer Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myServerName: RWCString`
Server Version

`myServerVersion: RWCString`

`mySystemLog: GlLog*`
Server Name

Constructors and Destructor:

`DsDoServer();`
BEGIN_PROLOG
NAME DsDoServer()
DESCRIPTION default constructor
RETURNS none
GLOBALS none
FILES none
END_PROLOG

`DsDoServer(const RWCString& serverName);`
Default constructor BEGIN_PROLOG

NAME DsDoServer(const RWCString &serverName)
DESCRIPTION normal constructor
RETURNS none
GLOBALS none
FILES none
END_PROLOG

```
~DsDoServer();
```

Overloaded constructor BEGIN_PROLOG
NAME ~DsDoServer
DESCRIPTION default destructor
RETURNS none
GLOBALS none
FILES none
END_PROLOG

Operations:

- saveOn

```
void saveOn(ostream& os) const;
```

Start up server BEGIN_PROLOG
NAME saveOn(ostream &os) const;
DESCRIPTION Global stream operator for DsShCommandBase objects.
RETURNS void
GLOBALS none
FILES none
END_PROLOG

- ShutDown

```
GlStatus ShutDown(void );
```

Default destructor BEGIN_PROLOG
NAME ShutDown (PHASE3)
DESCRIPTION shut down the server
RETURNS GlStatus
GLOBALS none
FILES none
END_PROLOG

- StartUp

  `GlStatus StartUp(void );`

  Shut down server BEGIN_PROLOG

  NAME StartUp (PHASE3)

  DESCRIPTION start up the server

  RETURNS GlStatus

  GLOBALS none

  FILES none

  END_PROLOG

5.3.4 es Class Category

5.3.4.1 Overview

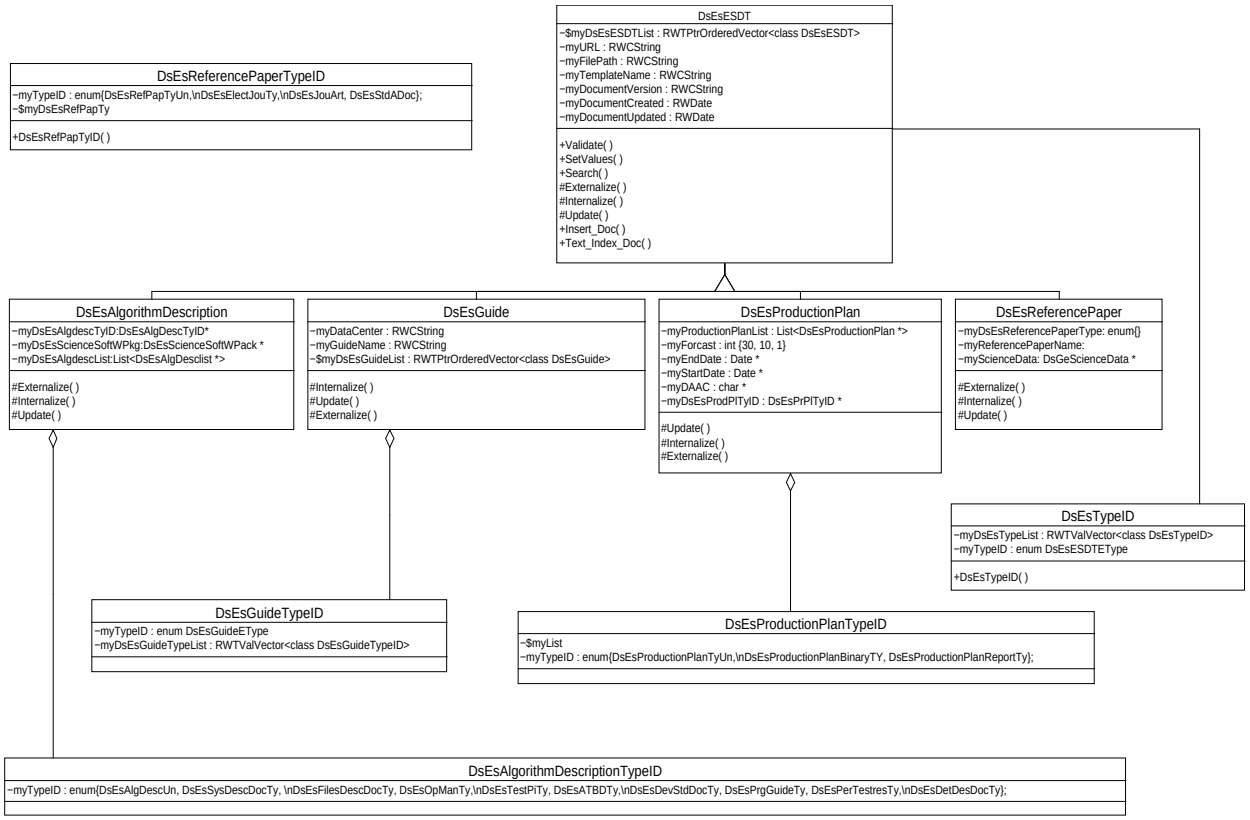


Figure 98 Main

5.3.4.2 es Classes

5.3.4.2.1 DsEsAlgorithmDescriptionTypeID Class

Overview:

Distributed Object: No This object represents the ID of the Algorithm Description type for the ESDT's.

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myTypeID: enum{DsEsAlgDescUn, DsEsSysDescDocTy,
\nDsEsFilesDescDocTy, DsEsOpManTy,\nDsEsTestPiTy,
DsEsATBDTy,\nDsEsDevStdDocTy, DsEsPrgGuideTy,
DsEsPerTestresTy,\nDsEsDetDesDocTy};
```

This attribute is used to distinguish between the different Algorithm descriptions.

Constructors and Destructor:

```
~DsEsAlgDescTyID();
```

This operation provides the desctructor for Type ID of the ESDT's Algorithm descriptions.

Operations:

- DsEsAlgDescTyID

```
DsEsAlgDescTyID(char *);
```

description.

- GetTypeID

```
GetTypeID();
```

This operation retrieves the Type ID of the Algorithm Description.

- SetTypeID

```
SetTypeID();
```

This operation sets the Type ID of the Algorithm description.

5.3.4.2.2 DsEsAlgorithmDescription Class

Overview:

Distributed Object: No This object represents the Algorthimic description for the ESDT's of document type.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsEsESDT

Attributes:

```
myDsEsAlgdescList:List<DsEsAlgDesclist *>:
```

```
myDsEsAlgdescList:
```

```
myDsEsAlgdescTyID:DsEsAlgDescTyID*:
```

```
myDsEsAlgdescTyID:
```

```
myDsEsScienceSoftWPkg:DsEsScienceSoftWPack *:
```

```
myDsEsScienceSoftWPkg:
```

Constructors and Destructor:

Operations:

- Externalize

```
Externalize();
```

- Internalize

```
Internalize();
```

- Update

```
Update();
```

5.3.4.2.3 DsEsESDT Class

Overview:

#include "ctpublic.h" #include "DsGeESDTBase.h" Class: DsEsESDT Distributed Object: No Description: This object represents the ESDT's of document type. SubSystemID: DDSRV

Export Control: Implementation

Inheritance Relationships:

Attributes:

myDocumentCreated: RWDate

define the attribute myDocumentCreated

myDocumentUpdated: RWDate

define the attribute myDocumentUpdated

myDocumentVersion: RWCString

define the attribute myDocumentVersion

myDsEsESDTList: RWTPtrOrderedVector<class DsEsESDT>

define the attribute _myDsEsESDTList

myFilePath: RWCString

description of attribute myDsEsTypeID DsEsTypeID myDsEsTypeID; description of attribute myDsEsCSDT DsEsCSDT * myDsEsCSDT; define the attribute myFilePath

myTemplateName: RWCString

define the attribute myTemplateName

myTemplateVersion: RWCString

define the attribute myTemplateVersion

myURL: RWCString

define the URL of the ESDT

Constructors and Destructor:

`DsEsESDT(RWCString URL, RWCString FilePath, RWCString TemplateName, RWCString TemplateVersion, RWCString DocumentVersion);`

DsEsESDT

description of operation Constructer BEGIN_PROLOG

NAME DsEsESDT

DESCRIPTION Constructer

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`DsEsESDT(const DsEsESDT& esdt);`

PrintList of objects inserted void PrintList(); Copy Constructer void DsEsESDT::PrintList() { cout << myDsEsESDTList.entries() << "entries\n" << endl; for (int i=0; i<myDsEsESDTList.length(); i++) cout << myDsEsESDTList[i] << endl; myDsEsESDTList.clearAndDestroy(); }

BEGIN_PROLOG

NAME DsEsESDT

DESCRIPTION Copy Constructer

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`~DsEsESDT();`

~DsEsESDT

Destructer BEGIN_PROLOG

NAME ~DsEsESDT

DESCRIPTION Destructer

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- Externalize

  ```
  void Externalize(void );
  ```

  Externalize

  Externalize the ESDT BEGIN_PROLOG

  NAME Externalize

  DESCRIPTION Phase 3

  RETURNS nothing

  GLOBALS none

  FILES none

  END_PROLOG

- Insert_Doc

  ```
  Insert_Doc();
  ```

- Internalize

  ```
  void Internalize(void );
  ```

  Internalize

  Internalize the ESDT BEGIN_PROLOG

  NAME Internalize

  DESCRIPTION Phase 3

  RETURNS nothing

  GLOBALS none

  FILES none

  END_PROLOG

- Search

  ```
  GlStatus Search(GlParameterList& keywords, GlParameterList&
  resultList);
  ```

  Search

  BEGIN_PROLOG

  NAME Search

  DESCRIPTION This member function makes search in sybase database given the search query parameters and returns the results.

  RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- SetValues

```
void SetValues();
```

SetValues Method BEGIN_PROLOG

NAME SetValues Method

DESCRIPTION Inputs the values for the data members

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- Text_Index_Doc

```
EcUtStatus Text_Index_Doc();
```

- Update

```
void Update(void );
```

Update

Update the ESDT BEGIN_PROLOG

NAME Update

DESCRIPTION Phase 3

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- Validate

```
void Validate(void );
```

Virtual Validate

description of operation Declaration of static members RWTPtrOrderedVector<DsEsESDT> myDsEsESDTList; BEGIN_PROLOG

NAME Validate (Phase 3 implementation)

DESCRIPTION Metadata Validation INPUTS GlParameterList& Metadata to validate OUTPUTS GlStatus& Status of validation

RETURNS none END_PROLOG

5.3.4.2.4 DsEsGuideTypeID Class

Overview:

Class: DsEsGuideTypeID Distributed Object: No Description: This object represents the ID of the Guide type. SubSystemID: DDSRV

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
myDsEsGuideTypeList: RWTValVector<class DsEsGuideTypeID>
```

description of attribute myDsEsGuideTypeList

```
myTypeID: enum DsEsGuideEType
```

description of attribute myTypeID

Constructors and Destructor:

```
DsEsGuideTypeID();
```

DsEsGuideTypeID

description of operation

```
~DsEsGuideTypeID();
```

~DsEsGuidetypeID

description of operation

Operations:

- GetDsEsGuideType

```
void GetDsEsGuideType(void );
```

GetDsEsGuideType

description of operation

- SetDsEsGuideList

```
void SetDsEsGuideList();
```

SetDsEsGuideList

description of operation

5.3.4.2.5 DsEsGuide Class

Overview:

Class: DsEsGuide Distributed Object: No Description: This object represents the Guide information for ESDT's. SubSystemID: DDSRV

Export Control: Implementation

Inheritance Relationships:

Inherits from DsEsESDT

Attributes:

`myDataCenter: RWCString`

description of attribute myDataCenter

`myDsEsGuideList: RWTPtrOrderedVector<class DsEsGuide>`

description of attribute myDsEsGuideTypeID DsEsGuideTypeID * myDsEsGuideTypeID; description of attribute myDsEsGuideList

`myGuideName: RWCString`

description of attribute myGuideName

Constructors and Destructor:

`DsEsGuide(RWCString DataCenter, RWCString GuideName);`

DsEsGuide : Normal Constructer BEGIN_PROLOG

NAME DsEsGuide.cxx

DESCRIPTION DsEsGuide class represents the Earth Science Data products of Guide type.

Functions provided for this class are:

Internalize Externalize Update

END_PROLOG

Declaration of Static Members RWTPtrOrderedVector<DsEsGuide> DsEsGuide::myDsEsGuideList; BEGIN_PROLOG

NAME DsEsGuide

DESCRIPTION Constructer

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`DsEsGuide(const DsEsGuide& guide);`

DsEsGuide : Copy Constructer BEGIN_PROLOG

NAME DsEsGuide

DESCRIPTION Copy Constructer

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

`~DsEsGuide();`

Destructer BEGIN_PROLOG

NAME ~DsEsESDT

DESCRIPTION Destructer

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

Operations:

- Externalize

  `void Externalize(void );`

  Externalize

  description of operation BEGIN_PROLOG

  NAME Externalize

  DESCRIPTION Phase 3

  RETURNS nothing

  GLOBALS none

  FILES none

  END_PROLOG

- Internalize

  `void Internalize(void );`

  Internalize

  description of operation BEGIN_PROLOG

NAME Internalize

DESCRIPTION Phase 3

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

- Update

```
void Update(void );
```

Update

description of operation BEGIN_PROLOG

NAME Update

DESCRIPTION Phase 3

RETURNS nothing

GLOBALS none

FILES none

END_PROLOG

5.3.4.2.6 DsEsProductionPlanTypeID Class

Overview:

Distributed Object: No This object represents the ID of the type of the Production Plan.

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
$myList: List<DsEsProductionPlanTypeID *>
```

List of currently active objects mainly used for fault recovery and memory management.

```
myList:
```

```
myTypeID:
enum{DsEsProductionPlanTyUn,\nDsEsProductionPlanBinaryTY,
DsEsProductionPlanReportTy};
```

Production Plan used to distinguish between different types of Production Plans.

Constructors and Destructor:

Operations:

- GetTypeID

  ```
  GetTypeID();
  ```

  This operation retreives the Type ID for the Production Plans of the ESDT's.

- SetTypeID

  ```
  SetTypeID();
  ```

  This operation sets the Type ID of the Production Plans for the ESDT's.

5.3.4.2.7 DsEsProductionPlan Class

Overview:

Distributed Object: No This object represents the Production Plan for the ESDT's of document type.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsEsESDT

Attributes:

```
myDAAC: char *

myDsEsProdPlTyID: DsEsPrPlTyID *

myEndDate: Date *

myForcast: int {30, 10, 1}

myProductionPlanList: List<DsEsProductionPlan *>

myStartDate: Date *
```

Constructors and Destructor:

Operations:

- Externalize

```
Externalize();
```

- Internalize

```
Internalize();
```

- Update

```
Update();
```

5.3.4.2.8 DsEsReferencePaperTypeID Class

Overview:

Distributed Object: No This object represents the ID of the Reference Papers of ESDT's.

Export Control: Implementation

Inheritance Relationships:

Attributes:

```
$myDsEsRefPapTy: List<DsEsRefPapTyID *>
```

List of the currently active objects. Mainly used for fault recovery and memory management.

```
myDsEsRefPapTy:
```

```
myTypeID: enum{DsEsRefPapTyUn,\nDsEsElectJouTy,\nDsEsJouArt,
DsEsStdADoc};
```

High level type ID used to distinguish between the main document types. Currently this includes Guides, Algorithm Descriptions, Reference Papers and Production plans.

Constructors and Destructor:

Operations:

- DsEsRefPapTyID

  ```
  DsEsRefPapTyID(char * name);
  ```

  Cosnstructor for the Reference paper Type ID.

- GetTypeID

  ```
  GetTypeID();
  ```

  This operation retrieves the Type ID for the Reference Paper.

- SetTypeID

  ```
  SetTypeID();
  ```

  This operation sets the Reference Paper Type ID.

5.3.4.2.9 DsEsReferencePaper Class

Overview:

Distributed Object: No This object represents the Reference Papers of Document type for the ESDT's.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsEsESDT

Attributes:

```
myDsEsReferencePaperList:List<DsEsreferncepaper *>:

myDsEsReferencePaperList:

myDsEsReferencePaperType: enum{}:

myDsEsReferencePaperType:
```

```
myReferencePaperName::
```

```
myReferencePaperName:
```

```
myScienceData: DsGeScienceData *:
```

```
myScienceData:
```

Constructors and Destructor:

Operations:

- Externalize

```
Externalize();
```

- Internalize

```
Internalize();
```

- Update

```
Update();
```

5.3.4.2.10 DsEsTypeID Class

Overview:

Class: DsEsTypeID Distributed Object: No Description: This object represents the ID of the type of the ESDT. SubSystemID: DDSRV

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myDsEsTypeList: RWTValVector<class DsEsTypeID>`

description of attribute myDsEsTypeList

`myTypeID: enum DsEsESDTEType`

description of attribute myTypeID

Constructors and Destructor:

`DsEsTypeID(void );`

DsEsTypeID

description of operation

`DsEsTypeID(char* );`

DsEsTypeID

description of operation

Operations:

- GetDsEsType

`void GetDsEsType(void );`

GetDsEsType

description of operation

- SetDsEsType

`void SetDsEsType(void );`

SetDsEsType

description of operation

5.3.5 gw Class Category

5.3.5.1 Overview

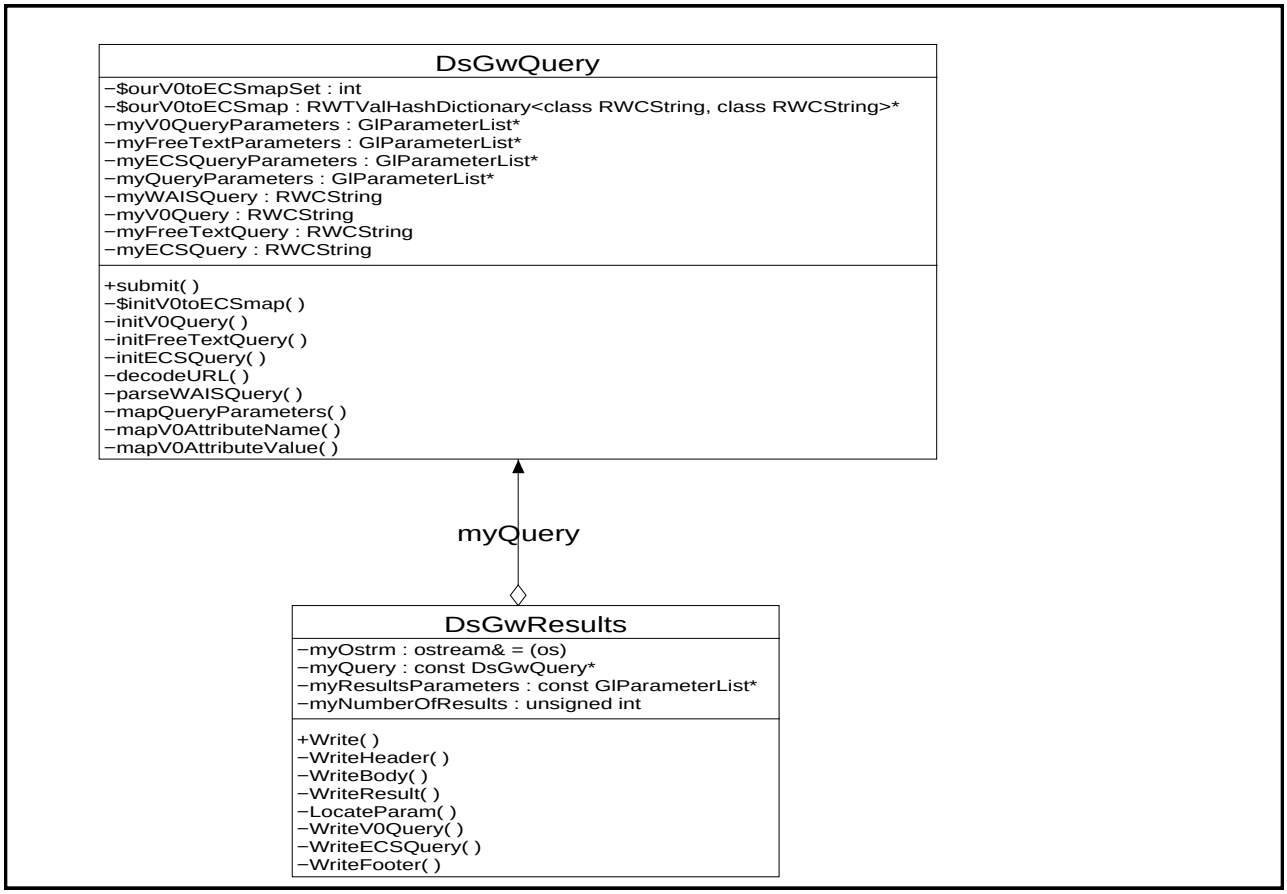


Figure 99 Main

5.3.5.2 gw Classes

5.3.5.2.1 DsGwQuery Class

Overview:

Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myECSQueryParameters: GlParameterList*`

Free text Paramters

`myECSQuery: RWCString`

String version of FreeText

`myFreeTextParameters: GlParameterList*`

V0 Query Parameters

`myFreeTextQuery: RWCString`

String version of V0Query

`myQueryParameters: GlParameterList*`

ECS Query Paramters

`myV0QueryParameters: GlParameterList*`

String version of ECSQuery

`myV0Query: RWCString`

Original query string

`myWAISQuery: RWCString`

`ourV0toECSmapSet: int`

static flag indicating whether ourV0toECSmap has been initialized

`ourV0toECSmap: RWTValHashDictionary<class RWCString, class RWCString>*`

static collection containing the valid base types

Constructors and Destructor:

`DsGwQuery();`

BEGIN_PROLOG

NAME DsGwQuery()

DESCRIPTION Default constructor

RETURNS void

GLOBALS none

FILES none

END_PROLOG

```
~DsGwQuery();
```

Default Constructor BEGIN_PROLOG

NAME ~DsGwQuery()

DESCRIPTION Default Destructor

RETURNS void

GLOBALS none

FILES none

END_PROLOG

Operations:

- decodeURL

```
GlStatus decodeURL(RWCString& url);
```

Level in tree defaults to top BEGIN_PROLOG

NAME decodeURL(RWCString &url)

TITLE Decode URL encoded strings

DESCRIPTION When URLs are transmitted from client to server, they are encoded to replace " " with "+" and other 'special' characters with their hex ascii equivalent using the convention %nn where nn is hex ascii value of the replaced character.

This private member function decodes the input string into a form which may be parsed by parseWAISQuery().

INPUTS <Optional> Encoded URL

OUTPUTS <Optional> Decoded URL

RETURNS GlStatus

GLOBALS none

FILES none

SYNOPSIS <Optional> Defined in HTTP 2.0 standard.

EXAMPLES <Optional> Encoded URL = "a+b+c+%41+%42+%43" Decoded URL = "a b c A B C"

Note: %41 - is the hex ascii value for the character A %42 - is the hex ascii value for the character B %43 - is the hex ascii value for the character C

NOTES <Optional> Makes use of the RWCRegexp class to match the regular expression "%[0-9a-fA-F][0-9a-fA-F]" in the input string and uses strtol in base 16 to find the equivalent ascii value.

The function expects 7-bit ascii input. It will return an error if the high bit is set.

DETAILS <Optional> none

REQUIREMENTS <Optional> none

END_PROLOG

- GetECSQuery

```
const RWCString& GetECSQuery() const;
```

Return V0Query string BEGIN_PROLOG
NAME GetECSQuery()
DESCRIPTION none
RETURNS myECSQuery
GLOBALS none
FILES none
END_PROLOG

- GetFreeTextQuery

```
const RWCString& GetFreeTextQuery() const;
```

Return ECSQuery string BEGIN_PROLOG
NAME GetFreeTextQuery()
DESCRIPTION none
RETURNS myFreeTextQuery
GLOBALS none
FILES none
END_PROLOG

- GetQuery

```
const GlParameterList* GetQuery() const;
```

Return FreeTextQuery string BEGIN_PROLOG
NAME GetQuery()
DESCRIPTION none
RETURNS myQueryParameters
GLOBALS none
FILES none
END_PROLOG

- GetV0Query

```
const RWCString& GetV0Query() const;
```

Return WAISQuery string BEGIN_PROLOG
NAME GetV0Query()
DESCRIPTION none

RETURNS myV0Query

GLOBALS none

FILES none

END_PROLOG

- GetWAISQuery

```
const RWCString& GetWAISQuery() const;
```

WAIS query string URL encoded. BEGIN_PROLOG

NAME GetWAISQuery()

DESCRIPTION none

RETURNS myWAISQuery

GLOBALS none

FILES none

END_PROLOG

- initECSQuery

```
void initECSQuery(GlParameterList& params, int level);
```

Level in tree defaults to top BEGIN_PROLOG

NAME initECSQuery(GlParameterList &params, int level)

DESCRIPTION Constructs myECSQuery RWCString based on the parameters in myECSQueryParameters. A string equivalent of the parameter list is useful when returning results to the client.

This private member function takes the input parameter list and defaults to level=0 (top). Level is incremented each recursive call when navigating the parameter list. The ECS DBMS Wrapper layer uses alternating levels in the parameter list to represent AND and OR clauses in the query. A simple test of level%2 can be used to determine if we are traversing an AND || OR sublist.

Note: All parameters are expected to be either GlParmType::PARAMETERLIST or GlParmType::STRINGP. No other parameter types are supported.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- initFreeTextQuery

```
void initFreeTextQuery(GlParameterList& params, int level);
```

Level in tree defaults to top BEGIN_PROLOG

NAME initFreeTextQuery(GlParameterList &params, int level)

DESCRIPTION Constructs myFreeTextQuery RWCString based on the parameters in myFreeTextParameters. A string equivalent of the parameter list is useful when returning results to the client.

This private member function takes the input parameter list and defaults to level=0 (top). Level is incremented each recursive call when navigating the parameter list. The ECS DBMS Wrapper layer uses alternating levels in the parameter list to represent AND and OR clauses in the query. A simple test of level%2 can be used to determine if we are traversing an AND || OR sublist.

Note: All parameters are expected to be either GlParmType::PARAMETERLIST or GlParmType::STRINGP. No other parameter types are supported.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- initV0Query

```
void initV0Query(GlParameterList& params, int level);
```

Initialize V0-ECS name map BEGIN_PROLOG

NAME initV0Query(GlParameterList &params, int level)

DESCRIPTION Constructs myV0Query RWCString based on the parameters in myV0QueryParameters. A string equivalent of the parameter list is useful when returning results to the client.

This private member function takes the input parameter list and defaults to level=0 (top). Level is incremented each recursive call when navigating the parameter list. The ECS DBMS Wrapper layer uses alternating levels in the parameter list to represent AND and OR clauses in the query. A simple test of level%2 can be used to determine if we are traversing an AND || OR sublist.

Note: All parameters are expected to be either GlParmType::PARAMETERLIST or GlParmType::STRINGP. No other parameter types are supported.

RETURNS void

GLOBALS none

FILES none

END_PROLOG

- initV0toECSmap

```
static GlStatus initV0toECSmap();
```

FreeText + ECS Parameters BEGIN_PROLOG

NAME initV0toECSmap()

DESCRIPTION Initialize V0-ECS name map. The name map is implemented for PHASE2 as RWTValHashDictionary. The V0 attribute name is used to key into the table and the ECS attribute name set to the value.

Note: This static member function should only be called once when the constructor for the first object is called. The static Boolean ourV0toECSmapSet ensures this function is called only once during the program execution.

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- mapQueryParameters

`GlStatus mapQueryParameters(GlParameterList& V0Params, GlParameterList& ECSParams);`

Parses input WAIS query BEGIN_PROLOG

NAME mapQueryParameters(GlParameterList &V0Params, GlParameterList &ECSParams)

TITLE Performs mapping of attribute name and value (PHASE3).

DESCRIPTION Maps V0->ECS name and value

INPUTS <Optional> myV0QueryParameters - Parameter list in V0 terms

OUTPUTS <Optional> myECSQueryParameters - Parameter list in ECS terms

RETURNS Glstatus

GLOBALS none

FILES none

SYNOPSIS <Optional> Each V0 parameter is visited and the equivalent ECS name and value discovered using mapV0AttributeName() and mapV0AttributeValue(). The parameter is inserted into the equivalent position in myECSQueryParameters parameter list.

EXAMPLES <Optional> none

NOTES <Optional> This member function is recursively called to handle sublists.

Strange Wierdness: The ECS DBMS wrapper layer uses the parameter description field as the operator. For all parameters this is set to "=" in the V0 parameter list and is copied into the ECS parameter list.

DETAILS <Optional> The non-recursive solution would be more complex to implement

REQUIREMENTS <Optional> none

END_PROLOG

- mapV0AttributeName

`RWCString mapV0AttributeName(RWCString& V0AttributeName);`

Ouput ECS parameter list BEGIN_PROLOG

NAME mapV0AttributeName(RWCString &V0AttributeName)

DESCRIPTION Maps V0->ECS name. Uses the RWTValHashDictionary to find the equivalent ECS name. For PHASE2, a 1:1 name mapping is sufficient.

PHASE3 - Integration with V0-ECS mapper object from CIDM

RETURNS RWCString

GLOBALS none

FILES none

END_PROLOG

- mapV0AttributeValue

```
RWCString mapV0AttributeValue(RWCString& V0AttributeValue);
```

Input V0 attribute name BEGIN_PROLOG

NAME mapV0AttributeValue(RWCString &V0AttributeValue)

DESCRIPTION Simply returns input V0AttributeValue. Why ? you may well ask:

For PHASE2 - We are manually populating the ECS database with valid values taken from V0 so no attribute value mapping will be required

For PHASE3 - We *may* need to map attribute values, and this function provides an integration point to call the CIDM V0-ECSMapper object.

RETURNS RWCString ECSAttributeValue

GLOBALS none

FILES none

END_PROLOG

- parseWAISQuery

```
GlStatus parseWAISQuery();
```

Encoded URL String BEGIN_PROLOG

NAME parseWAISQuery()

TITLE Parses an input V0 WAIS formatted query

DESCRIPTION Parses input WAIS query

INPUTS <Optional>

WAIS formatted query string of the form:

a=1 and b=2 and c=3 and d

OR

a=1 or b=2 or c=3 or d

Note: The "=" operator a query string is used to determine if the term is a keyword=value pair. All other terms are assumed to be FreeText query terms.

OUTPUTS <Optional> GlParameterList equivalent of the input WAIS formatted query. The ECS production rules for and/or conditions are used to structure the query parameters. The keyword and value names are still in their input form and have not been mapped to their ECS equivalents

RETURNS GlStatus

GLOBALS none

FILES none

SYNOPSIS <Optional> Uses the RWCTokenizer to tokenize the input string on whitespace. This assumes that the input query string has been decoded.

while (!(token=next()).isNull()) { if(keword=value) { if(ANDstatement) add parameter at this level if(ORStatement) create sublist and add parameter } if(freetext) { add parameter to freetext list } }

EXAMPLES <Optional> INPUT Query String WAIS Query (a=1 and b=2 and c=3 and d)

OUTPUT Parameter Lists myV0QueryParameters[a(1)b(2)c(3)] myFreeTextParameters[FreeText(d)]

NOTES <Optional> The parser has *not* been tested on mixed and/or input WAIS queries as these are not generated by the Release A Client.

DETAILS <Optional> This code is intended to parse V0 formatted query strings and is not intended to be a general WAIS query parser.

REQUIREMENTS <Optional> none

END_PROLOG

- submit

`GlStatus submit(char* query);`

Default Destructor BEGIN_PROLOG

NAME submit(char *query)

TITLE V0->ECS query translation

DESCRIPTION Translate V0 query into ECS query

INPUTS <Optional> WAIS formatted query string URL encoded

OUTPUTS <Optional> Places translated ECS query in myQueryParameters

RETURNS GlStatus

GLOBALS none

FILES none

SYNOPSIS <Optional> Controls the call sequence to private member functions:

decodeURL parseWAISQuery mapQueryParameters initECSQuery

Returns status to calling function.

EXAMPLES <Optional> submit("Sensor=AVHRR+or+FRED")

NOTES <Optional> This member function controls the translation of an V0 query into ECS terms. It does *not* submit the query to the ECS DBMS Wrapper layer. That is the responsibility of the calling function.

DETAILS <Optional> Calls initECSQuery to construct the equivalent V0 query string.

REQUIREMENTS <Optional> none
END_PROLOG

5.3.5.2.2 DsGwResults Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

`myNumberOfResults: unsigned int`
Results parameter list

`myOstrm: ostream&`

`myQuery: const DsGwQuery*`
Output stream to direct results to

`myResultsParameters: const GlParameterList*`
Associated query object

Constructors and Destructor:

`DsGwResults(void );`

`DsGwResults(ostream& os);`
Default constructor

`~DsGwResults(void );`
Output directed to ostream BEGIN_PROLOG
NAME ~DsGwResults()
DESCRIPTION Default destructor
RETURNS void
GLOBALS none
FILES none

END_PROLOG

Operations:

- LocateParam

  ```
  const GlParameter* LocateParam(const GlParameter* params, const char* name, int typ) const;
  ```

  The number of this hit in list. BEGIN_PROLOG

  NAME LocateParam

  DESCRIPTION Locates a named parameter in list. Only locates parameters in the current list and does not desend into sublists. Will return NULL if parameter can not be found or is not of the expected type.

  RETURNS const GlParameter *

  GLOBALS none

  FILES none

  END_PROLOG

- SetQuery

  ```
  void SetQuery(const DsGwQuery* query);
  ```

  Location of results parameter BEGIN_PROLOG

  NAME SetQuery

  DESCRIPTION Set the associated query object

  RETURNS void

  GLOBALS none

  FILES none

  END_PROLOG

- SetResultsParameters

  ```
  GlStatus SetResultsParameters(const GlParameterList* params);
  ```

  Default destructor BEGIN_PROLOG

  NAME SetResultsParameters

  DESCRIPTION Set the results parameter list

  RETURNS GlStatus

  GLOBALS none

  FILES none

  END_PROLOG

- WriteBody

```
GlStatus WriteBody(const GlParameterList* params) const;
```

Write out the HTML header BEGIN_PROLOG

NAME WriteBody

DESCRIPTION Write out the body of the results

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- WriteECSQuery

```
GlStatus WriteECSQuery() const;
```

Write out the V0 query string BEGIN_PROLOG

NAME WriteECSQuery

DESCRIPTION Write out the ECS query string

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- WriteFooter

```
GlStatus WriteFooter() const;
```

Write out the ECS query string BEGIN_PROLOG

NAME WriteFooter

DESCRIPTION Write out the HTML footer

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- WriteHeader

```
GlStatus WriteHeader() const;
```

Number of results in result list BEGIN_PROLOG

NAME WriteHeader

DESCRIPTION Write out the HTML header

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- WriteResult

```
GlStatus WriteResult(const GlParameter* result, int hit) const;
```

BEGIN_PROLOG

NAME WriteResult

DESCRIPTION Write out a single HTML result. Expects ECS results format to be returned from ECS DBMS Wrapper layer. Will fail if a required parameter name can not be located or if parameter is of an unexpected type.

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- WriteV0Query

```
GlStatus WriteV0Query() const;
```

BEGIN_PROLOG

NAME WriteV0Query

DESCRIPTION Write out the V0 query string

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

- Write

```
GlStatus Write();
```

Set the associated query object BEGIN_PROLOG

NAME Write

DESCRIPTION Write HTML formatted results list.

Note: If the DsGwDEBUG flag is set, make sure the "Content-type: text/plain" is output first. This will prevent your Web browser having a heart attack when the debug trace is returned with "Content-type: text/html".

RETURNS GlStatus

GLOBALS none

FILES none

END_PROLOG

5.3.6 se Class Category

5.3.6.1 Overview

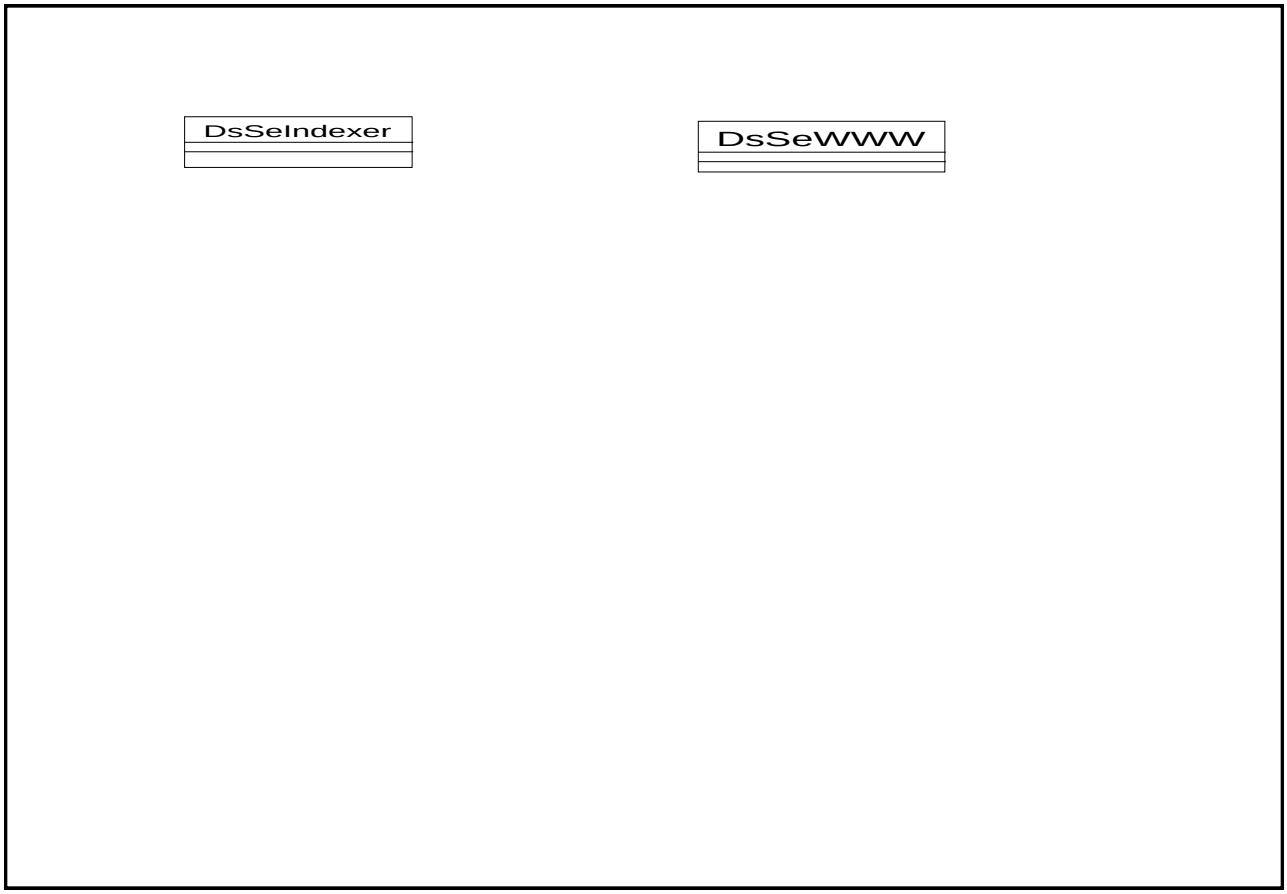


Figure 100 Main

5.3.6.2 se Classes

5.3.6.2.1 DsSeIndexer Class

Overview:

Distributed Object: No This object represents the COTS Technology for free text Indexing.

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

5.3.6.2.2 DsSeWWW Class

Overview:
Distributed Object: No

Export Control: Implementation

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

5.3.7 sv Class Category

5.3.7.1 Overview

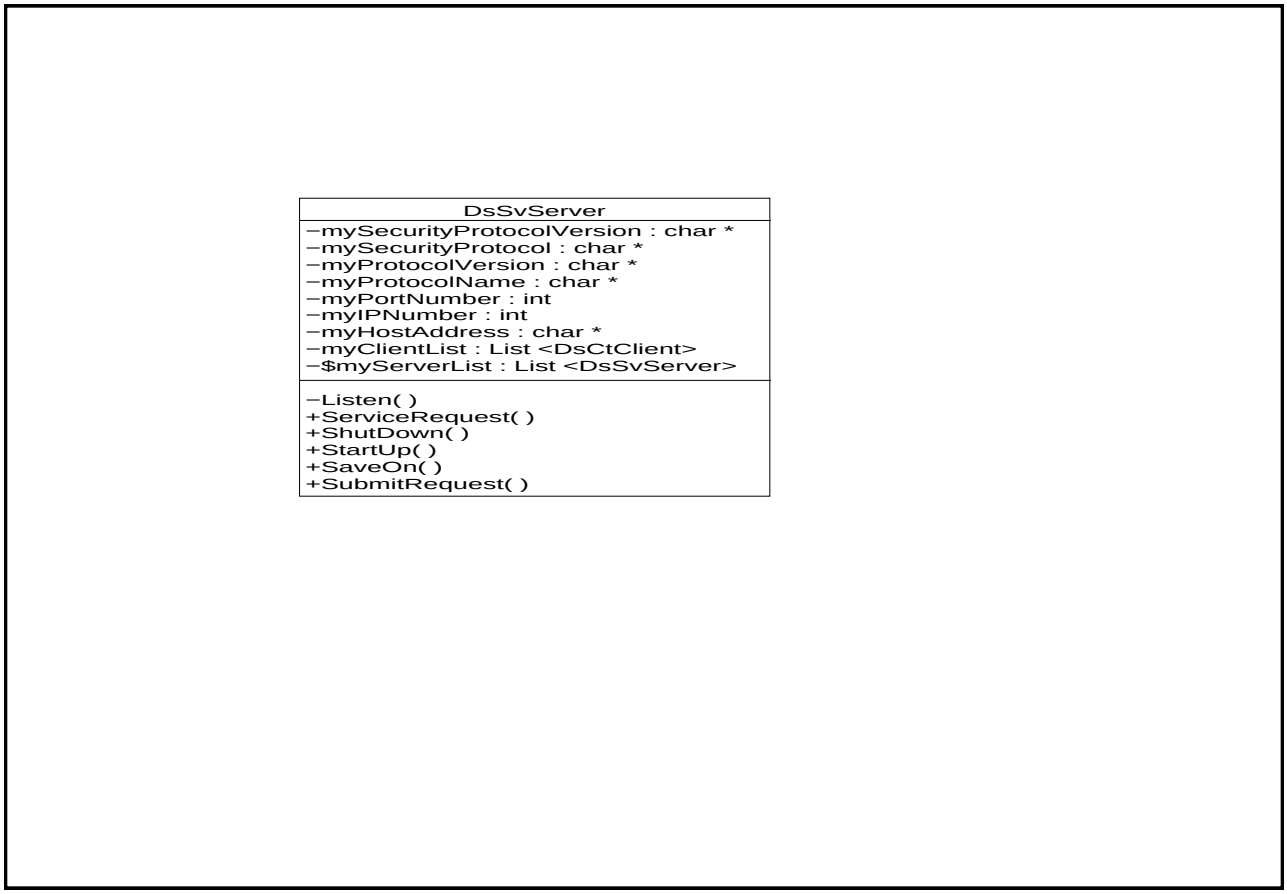


Figure 10 Main

5.3.7.2 sv Classes

5.3.7.2.1 DsSvServer Class

Overview:

Distributed Object: No The Object represents the server object for Document Data Server.

Export Control: Implementation

Inheritance Relationships:

Inherits from DsDoServer

Attributes:

`$myServerList: List <DsSvServer>`

List of currently active servers. Used for fault recovery and memory management.

`myClientList: List <DsCtClient>`

List of the active clients being serviced by the server.

`myHostAddress: char *`

`myIPNumber: int`

IP number of the server.

`myPortNumber: int`

TCP/IP port number on the host to monitor for incoming HTTP requests.

`myProtocolName: char *`

Name of the communication protocol supported by the server.

`myProtocolVersion: char *`

Version of the communication protocol supported by the server.

`mySecurityProtocolVersion: char *`

Version of the security protocol used by the server.

`mySecurityProtocol: char *`

Name of the security protocol used by the server. Needs to be compatable with the client security protocol for a secure connection.

`myServerList:`

Constructors and Destructor:

`~DsDoServer();`

Object destructor - destroy associated client objects

Operations:

- DsDoServer

`DsDoServer();`

Object constructor - no specific implementation

- Listen

  `GlStatus Listen();`

  Monitors the TCP/IP port for incomming requests.

- SaveOn

  `void SaveOn(ostream& os);`

  Default destructor

  DESCRIPTION Global stream operation for DsShCommandBase objects.

- ServiceRequest

  `GlStatus ServiceRequest();`

  Called by client to execute a service request.

- ShutDown

  `GlStatus ShutDown();`

  Shut down of an active server. Currently active clients will have their associated service requests aborted.

- StartUp

  `GlStatus StartUp();`

  Start up server. Return status to calling object indicating the server is ready to service incoming requests.

- SubmitRequest

  `SubmitRequest();`

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 DDSRV Document Insert

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. 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 file 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.

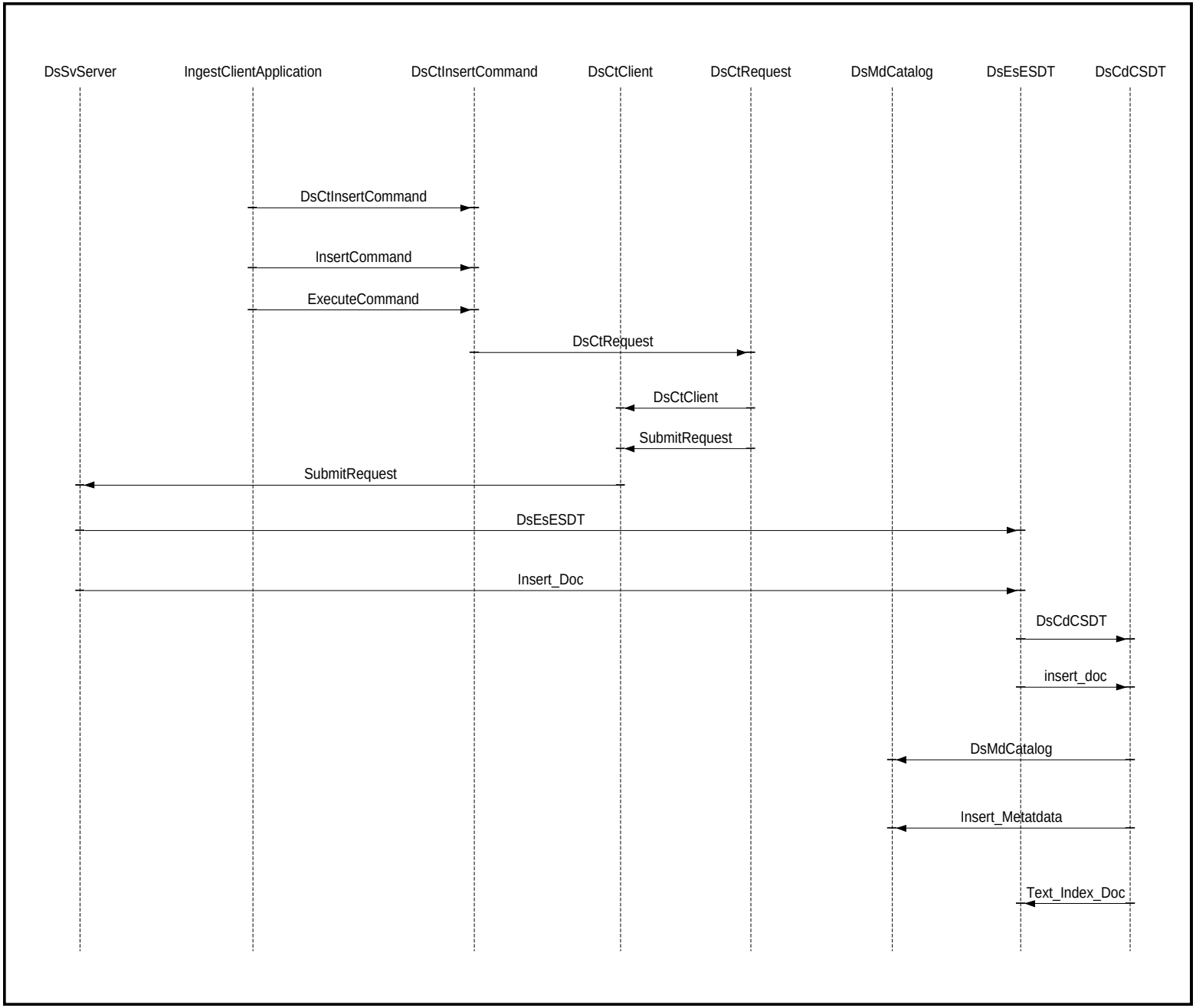


Figure 102 DSR V Document Insert

5.4.2 DDSRV search

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.

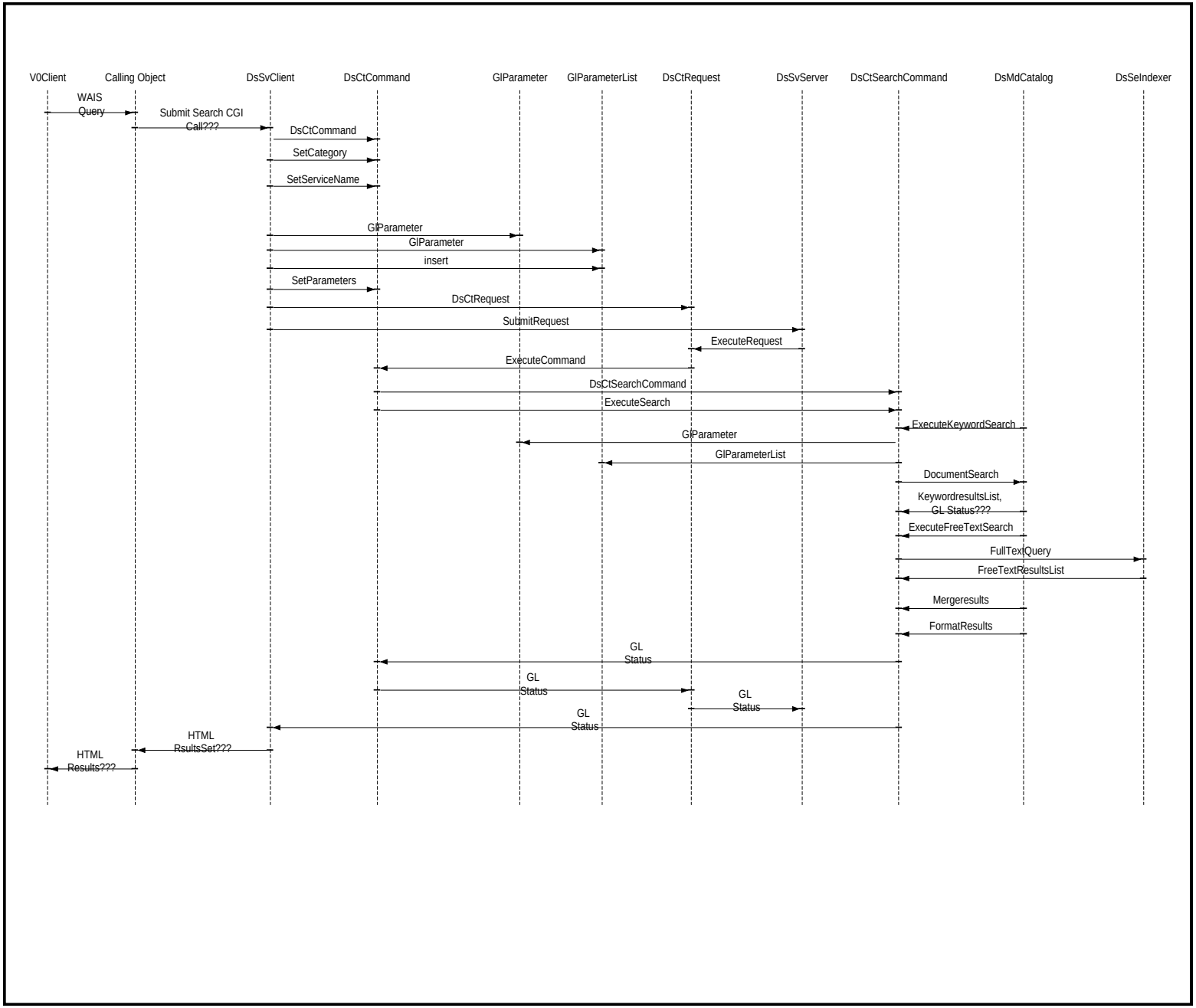


Figure 103: DSR V search

5.4.3 DDSRV Acquire Dynamic Model

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.

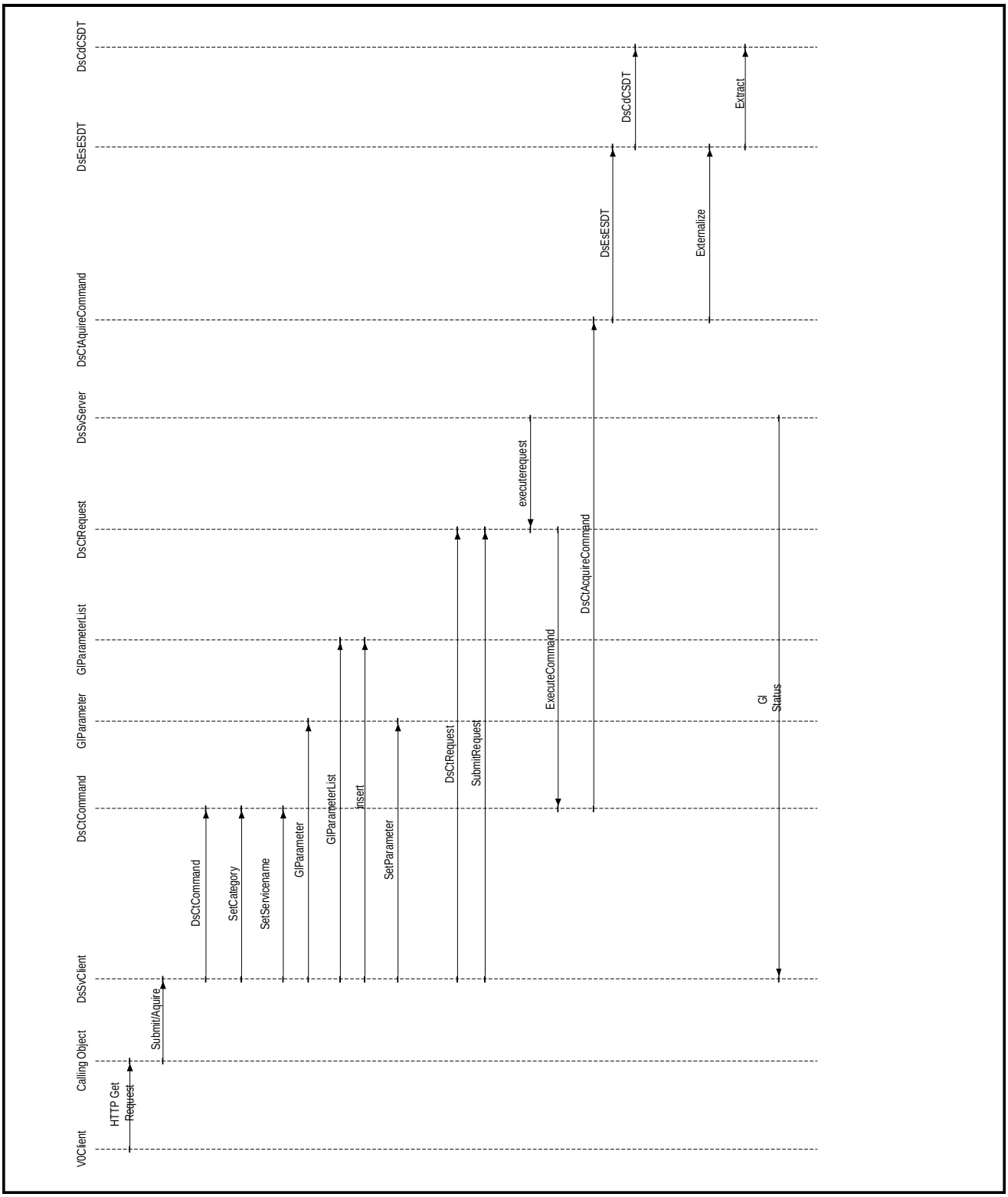


Figure 104: DSR V Acquire Dynamic Model

5.4.4 DDSRV Fault Scenario Dynamic Model

5.4.4.1 Summary

In this scenario, during the execution of a service request by the DDSRV, 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 DDSRV 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.

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 DDSRV 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.

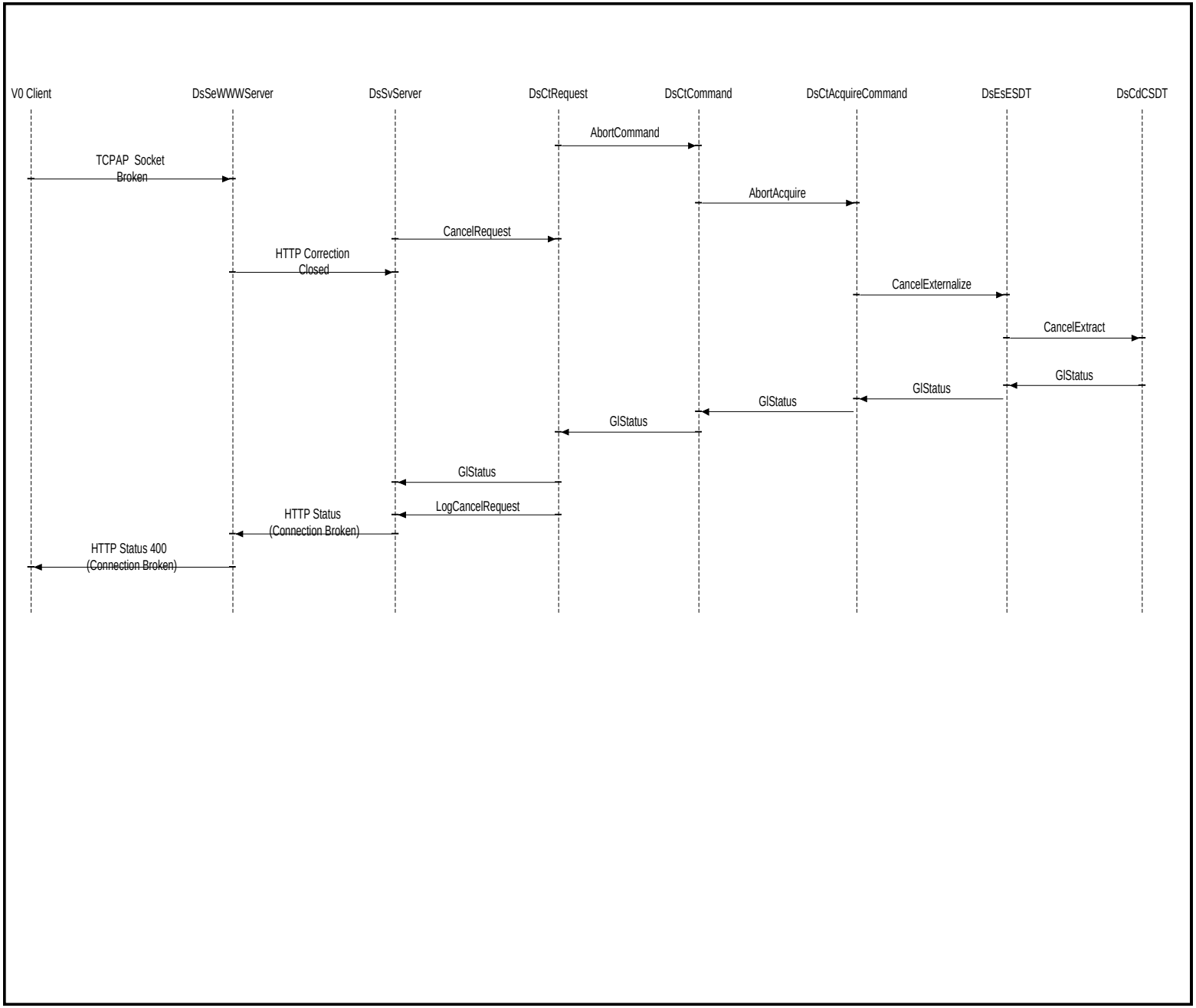


Figure 10-DSR V Fault Scenario Dynamic Model

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 HTTP access to document services, and full-text indexing and searching of documents.	COTS

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

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

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

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

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

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

COTS products

Netscape Enterprise Server

TOPIC by Verity, Inc.

ECS white paper references

N/A

5.5.2 DDSRV CI Processes

The Document Data Server is implemented with 5 software products; 4 processes and 1 client library.

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

DDSRV CSDT

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 the COTS product, Netscape Enterprise server, and of classes from the following CSC :

DDSRV Search Engine

5.5.2.3 Text Search Engine Process

The Text Search Engine Process provides the text index and searching capabilities, and consists of the COTS product, TOPIC by Verity, Inc., and of classes from the following CSC:

DDSRV Search Engine

5.5.2.4 HTTP Gateway Process

The HTTP Gateway Process provides the interface translation for requests from WWW clients. This process translates the queries recieved from clients from WAIS queries over HTTP, to the distributed client objects in the DDSRV Client applications library. This process consists of softare from the following CSCs “

DDSRV Search Engine

DDSRV Client CSC

5.5.2.5 Client Applications Library

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:

1) 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.

2) 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

10. DRPHW - Data Repository HWCI

Data Repositories (DRs) are the hardware components that store and maintain data and metadata permanently. Different technologies can be used to instantiate DRs 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

The DR hardware of the data server is designed for expandability and evolvability. The expandability is very important in the light of 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. The DR storage hardware differs at the large and small DAACs. A single DR may have heterogeneous storage elements within it in order to best accommodate DAAC-specific performance requirements.

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 periodically updated document reviews existing secondary storage technology solutions with the purpose of outlining technology use and insertion strategy for the ECS Data Server Permanent Data Repository over the Releases of ECS Project. The document reflects both past and ongoing analyses of the project requirements and the storage technologies market. (Note: This WP is considered to be competition sensitive and is not available for general review.) 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. 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 Arrays of Inexpensive Disks (RAID) pools and network attached Automated 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.

"Illustra Use Options" Trades. This study investigated the implementation options between the DDSRV and the SDSRV introduced by use of Illustra in the DRPHWCI for the Release B SDSRV. The trade examined several partial mergers of functionality, and the potential for a complete merger of the two servers in Release B. The trade concluded that the storage and maintenance of the documents and their metadata can be allocated to the Illustra server on the DRPHWCI with minimal impact. By reducing the number of Illustra instances in the Data Server, this provides a cost benefit to Release B. The details of the trade are presented in section 12 of this document, "DDSRVHW - Document Data Server HWCI".

10.1.2Scalability, Evolvability, and Migration in 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. Horizontal scaling, in this case, means that multiple DR hosts may be used to managed one or more ATLs with the possibility of multiple storage technologies of varying form factors. The physical configuration at each DAAC and the specific performance requirements at each site determines the host to device distribution.

Increasing Permanent Storage Capacity - 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 evolutions 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. 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.

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 (WKSHW). By enlarging Working Storage a reasonable write buffer can be maintained in the higher tiers of the architecture, thereby keeping the deep archive accesses down to a manageable level. Effectiveness of this approach will vary by site due to access patterns on the data stored in each repository.

The deep archives can also be scaled 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. 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 due to helical scan drive head spin up, a higher transfer rate over the length of a sizable file compensates for this time loss. Conversely, smaller files will either have to be blocked together for storage to realize a throughput benefit, or placed on a storage medium, such as IBM 3590 (NTP) linear tape, that imposes less penalty on access, even if the throughput rate is comparatively less. Helical scan and linear tape drive both have limitations that make exclusive use of either technology design inefficient. Helical scan technologies notoriously have higher maintenance costs associated with them, while linear technologies have significantly less storage density than comparable form factor helical media.

The baseline design for Release B is to use helical scan and/or linear tape solutions for the deep archives separately or in a mixed fashion where access patterns and storage densities dictate.

10.2 HWCI Structure

The sizing of Release B Data Repository components hardware was done using a combination of dynamic and static modeling and based on the assumptions outlined below.

a. AHWGP Technical Baseline (February 1996) data was used for the GB/day flow estimates. The product data was assumed to cumulate over the time period 9/1/1997 through 6/1/1999.

b. V0 PDR Baseline holdings based on quantities for the data sets in called out in the Technical baseline for the Release B period.

c. DAAC to DAAC flows are as published on March 3, 1996

The FSMS host platform was selected on the basis of FSMS/platform compatibility, available memory cache 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 Release B data as well that associated with the V0 data sets identified for migration within Release B. The sizing data also included nominal metadata sizing for documents associated with the Release B and V0 data sets, and the sizing for the documents themselves. 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

10.2.1 HWCI Connectivity

The Data Server's servers and workstations will be directly connected to the DAAC FDDI network and HiPPI fabric. 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 for a general description of DAAC networks.)

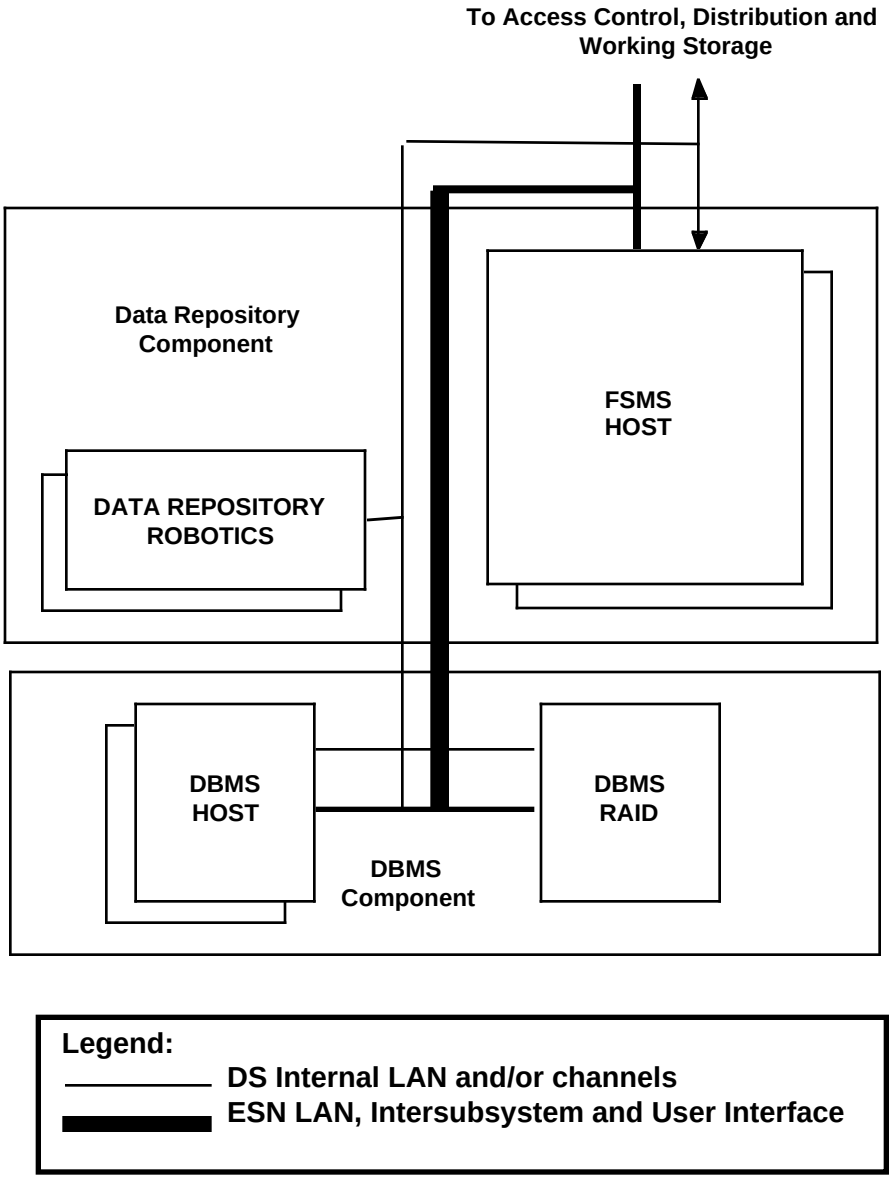


Figure 10.2-1. Data Repository HWCI Block Diagram

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 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. Technology has not yet reached this level, although many vendors and research labs are working on products/approaches in this area. Therefore, at this time Release B is not baselining the use of Network Attached Storage technologies, but we are keeping the S/W and H/W architectures designed to support the insertion of this technology. Logical separation of control and data paths will be maintained to allow for physical separation.

10.2.3 Failover and Recovery Strategy

The robotic units chosen for Release A are baselined at this point to continue into Release B at some sites based on each site's access patterns and required storage densities. Sites with high access and throughput requirements will require augmentation by higher speed multi-technology (helical and liner tapes of the same form factor), non-multimedia ATLs. Release A baselined a single archive robotic arm for each site. In Release B, the larger sites continuing with this technology will have a second arm installed. In the event of failure in a multi-armed configuration, the non-functioning arm is taken. Access to the associated storage occurs via the second arm in a degraded mode of operation.Remedial maintenance will then need to be scheduled by the site’s M&O staff for the failed arm. At the smaller sites that are configured with one arm, a failure condition will necessitate a repair to be performed as soon as practically possible. The EMASS robotics are field proven in Europe and the U.S. as having extremely high reliability. This fact, coupled with the non-trivial cost of the robotics, supports a one arm robotic solution at the smaller sites.

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. Multiple towers in larger Release B sites.
	Magnetic Tape Drives	Linear tape was baselined for Release A. Helical scan tape is planned for Release B.
	SMP Server Workstation (FSMS Host)	Server with Multiprocessor Capabilities. Dual processors were baselined in Release A. By CDR-B, sufficient EDF field experience will be used to determine how many more processors (if any) will be brought into the Release B configuration at the sites.
Data Base Management System (DBMS) Repository	SMP DBMS Server	Server with Multiprocessor Capabilities. Dual processors were baselined in Release A. By CDR-B, sufficient EDF field experience will be used to determine how many more processors (if any) will be brought into the Release B configuration at the sites.
	RAID (host attached)	DBMS RAID. The quantities specified Release A basically remain stable in Release B. (EDC is the exception in that a substantially larger data base is required.)

In Release A, only a small quantity of magnetic drives were required in the archive to reach performance requirements. Given this situation, the approach was to double the calculated required number of drives and have the redundant drive set provide the failover capability for the primary set. In Release B, because the number of required drives is much higher, one or two drives will be added for spares to reach RMA requirements.

The SMP Server workstations serving as FSMS hosts will be configured identically but with smaller number of CPU chips and share a single disk bank to supply failover/recovery capability. One of the workstations will be configured as primary, the other, with all peripheral connections but half the estimated processing power, as a cold standby secondary. At some sites, access requirements and server loads may require both hosts to function with separate primary and shared secondary responsibilities. Each host would be equipped with sufficient capacity to provide degraded mode operations in the event of a reported failure by one of the hosts. The active workstation will maintain the current FSMS configuration files on disk. If a failure on a primary workstation is reported by a MSS 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 MSS. During a failover the backup workstation will read the system status files from the shared disk and begin operation as primary.

Similarly, two SMP DBMS host machines in a cold standby configuration will share a common disk pool.

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 documents.

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, AM-1, and SAGE III missions.

12.1.1 Key Trade-off Studies and Prototypes

"Portable File Formats" TRADE The trade study analyzed 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.

"Illustra Use Options". This study investigated the implementation options introduced by use of Illustra in the DRPHWCI for the Release B SDSRV. Options presented by the use of Illustra by both the DDSRV and the SDSRV were to implement one or more partial solutions, or a complete integration of SDSRV and DDSRV

The Partial Merger options were as follows :

- Migrate Document Metadata with Science Metadata
- Merge Text Search Engine with Keyword Engine
- Merge Document Repository with Keyword Engine

The Complete Merger option was to fully integrate the DDSRV and SDSRV. Options and their trade results are as follows :

• <u>Partial Merger : Migrate Document Metadata with Science Metadata</u> : This trade evaluated the benefits/costs of migrating document keyword database with Science Data keyword database from Sybase to Illustra

Benefits

∑ Maintains "merged" schema implemented in Release A

∑ Application changes contained within DBMS wrappers

Impacts

∑ Migration of 4GL for DDSRV

Recommendation

∑ Continue to maintain merged SDSRV and DDSRV schema, migrate Document Metadata along with Science Metadata from Sybase to lllustra in Release B.

• <u>Partial Merger : Merge Text Search Engine with Keyword Engine</u> : This trade evaluated the potential of replacing the Release A Text Search Implementation with Illustra Datablades for Text

Search. The Release A text search implementation uses the Verity Developers Toolkit API to access the TOPIC text search engine. The Illustra Text Datablades provide a opportunity to encapsulate the text search in the DBMS wrapper layer.

Benefits

- ∑ Encapsulates Text Search to DBMS wrappers

Impacts

- ∑ Requirements not met by Text Search Datablades
- ∑ TOPIC Datablade not available until Sept/Oct 96
- ∑ Rework of Release A DBMS wrappers and DDSRV core

Recommendation

- ∑ Continue with Release A implementation, interface with the TOPIC Verity Developers Toolkit API.

• Partial Merger : Merge Document Repository with Keyword Engine: This trade evaluated the potential of consolidating the Document Repository from DDSRVHWCI to SDSRV Keyword Database on DRPHWCI

Benefits

- ∑ Reduces number of Illustra instances from 2 to 1 in DSS
- ∑ Simplifies ingest and maintenance
- ∑ Enables implementation of “WebExplode” feature for evolvability

Impacts

- ∑ Document transaction rates, I/O rates and data volumes are modest in comparison to SDSRV
- ∑ Rework of document ingest

Recommendation

- ∑ Co-locate Document Repository with Keyword Database on DRPHWCI

• Complete Merger : Fully Integrated DDSRV and SDSRV: This trade evaluated the potential of completely integrating the DDSRV with SDSRV.

Benefits

- ∑ Evolutionary enhancement - Single, unified DSS interface for ESDT and related documents

Impacts

- ∑ Public Classes for SDSRV and DDSRV would be changed
- ∑ Backend core processes would also require redesign

Recommendation

- ∑ Reserve complete integration for evolvability release
- ∑ Continue with current implementation of separate interface classes for documents and science data

∑ Implement results from Partial Merger trades

12.1.2 Scalability, Evolvability and Migration to Future Releases

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. The Release B requirements have not changed the Release A approach to hardware and sizing. In the future, 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-004-002 for a general description of DAAC networks.)

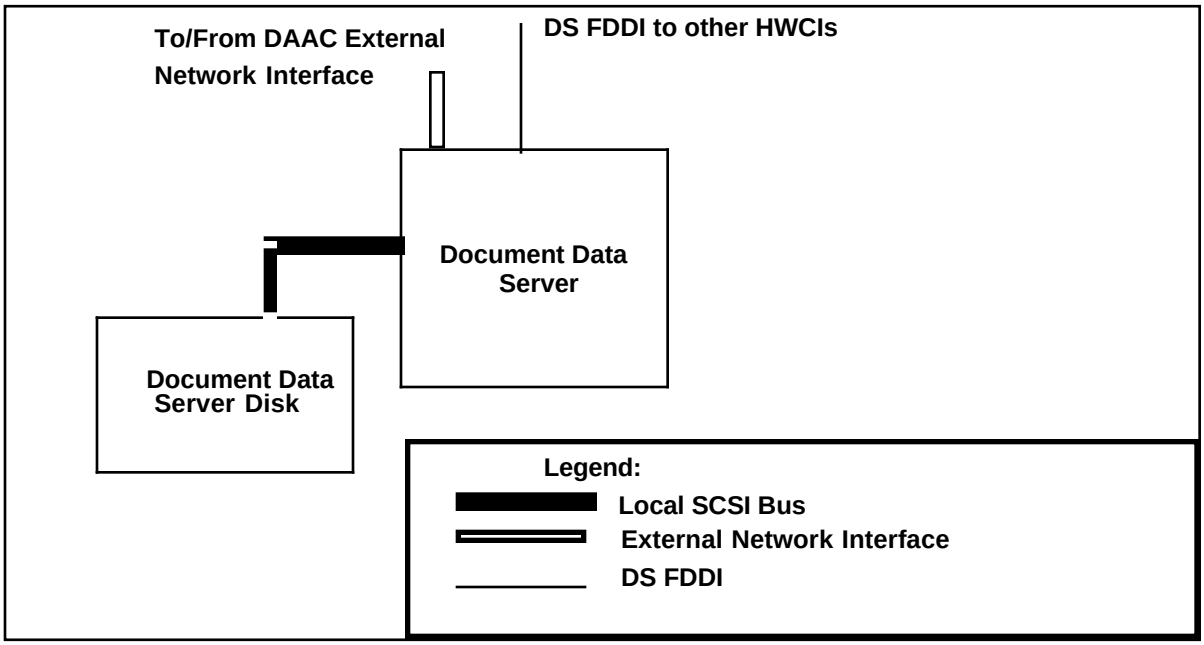


Figure 12.2-1. Document Data Server HWCI Block Diagram

12.2.2 HWCI Components

A 2 CPU SMP server was selected based upon operational experience with the EDF EDHS. A 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, AM-1, SAGE III missions, and for the V0 data sets identified for migration. Sizing for document metadata was based on the 2.0 Core metadata baseline. Growth was based on the phased migration of V0 data sets and the TRMM, AM-1, and SAGE III data product collections acquired during Release B operations.

Table 12.2-1. Access Control & 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 B, 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 A. Requirements Trace

I

Table A-1. DSS Release B Requirements Trace (1 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00070	The SDSRV CI shall accept Service Requests from the Data Processing subsystem and, as a result, provide access to Data for the purpose of reprocessing.	DsClRequest
S-DSS-00115	The SDSRV CI shall accept Search Status Requests for a specified active Search Request and, if requested, provide all Search Results accumulated for that Search Request.	DSS
S-DSS-00116	The SDSRV CI shall accept Search Status Requests for a specified active Search Request and, if requested, provide all Search Results accumulated since the last Search Status Request for that Search Request.	DSS
S-DSS-00180	The SDSRV CI shall accept and process Data Requests for Data Products that are produced on demand using the resources available to the Data Server.	DsGeECSDataProduct
S-DSS-00200	The SDSRV CI shall provide the capability for a user to delete their own queued Data Request.	DsClRequest
S-DSS-00210	The SDSRV CI shall provide operations staff the capability to update the Priority Information for a queued Service Request.	DsClRequest
S-DSS-00215	The SDSRV CI shall provide operations staff the capability to modify any field in a queued Service request.	DsClRequest
S-DSS-00230	The SDSRV CI shall provide users the capability to cancel their own Service Requests.	DsClRequest
S-DSS-00240	The SDSRV CI shall determine which Data Requests require post-retrieval processing.	DsGeESDT
S-DSS-00250	The SDSRV CI shall provide an application program interface for the submission of Service Requests.	Client CSC
S-DSS-00260	The SDSRV CI shall provide an application program interface for the submission of requests for administrative services.	Client CSC
S-DSS-00264	The SDSRV CI shall provide an application program interface which permits DAAC operations staff to link special subsetting capabilities into a Science Data Server.	DsGeDynamicLibrary
S-DSS-00270	The SDSRV CI shall accept and process Data Requests for Repaired Orbit Data.	DsGeESDT
S-DSS-00280	The SDSRV CI shall accept and process Data Requests for Attitude Data.	DsGeESDT
S-DSS-00290	The SDSRV CI shall accept Suspend Requests to suspend processing a client session.	DsClESDTReferenceCollector
S-DSS-00300	The SDSRV CI shall accept Resume Requests to resume processing of a client session.	DsClESDTReferenceCollector

Table A-1. DSS Release B Requirements Trace (2 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00310	The SDSRV CI shall provide the capability for authorized clients to submit Service Requests batch mode.	DsClRequest
S-DSS-00320	The SDSRV CI shall notify clients that issue Cancellation Requests that the associated Service Request has been canceled or the associated Service Request was completed.	DsClRequest
S-DSS-00330	The SDSRV CI shall record Request Identifiers to be used for accounting purposes.	DsSrRequestBase
S-DSS-00331	The SDSRV CI shall record the User Identifier of the science investigator associated with a Service Request, to be used for accounting purposes.	DsSrClient
S-DSS-00332	The SDSRV CI shall record the amount of user storage associated with a science user, to be used for accounting purposes.	DsUzDiskCostB
S-DSS-00333	The SDSRV CI shall record the amount of connect time associated with a science user, to be used for accounting purposes.	DsSrConnectionMaker
S-DSS-00340	The SDSRV CI shall record the level of CPU utilization for each Service Request to be used for accounting.	DsUzCPUCostB
S-DSS-00350	The SDSRV CI shall record the level of I/O utilization for each Service Request to be used for accounting.	DsUzIOCostB
S-DSS-00360	The SDSRV CI shall record, for accounting purposes, a fixed personnel cost for Service Requests requiring interaction with operations staff.	DsUzFixedCostB
S-DSS-00370	The SDSRV CI shall record a archival storage cost based on the number of bytes stored, to be used for accounting.	DsUzArchiveCostB
S-DSS-00375	The SDSRV CI shall associate User Accounting Information with client sessions.	DsSrClient
S-DSS-00376	The SDSRV CI shall provide User Accounting Information to the SMC.	EcPfManagedServer
S-DSS-00377	The SDSRV CI shall support operations staff in the creation of utilization reports, and operations staff shall distribute them on a periodic basis to a predefined list of report recipients.	MSS
S-DSS-00378	Operations staff shall be able to distribute SDSRV utilization reports electronically or in hard copy or on electronic media.	SDSRV CI
S-DSS-00400	The SDSRV CI shall accept pricing information, based on disk, CPU and media utilization, from CSMS.	DsUzCostB
S-DSS-00410	The SDSRV CI shall provide actual cost information by the completion of a Service Request.	DsUzResourceCostB
S-DSS-00420	The SDSRV CI shall record the amount of media utilized for a Distribution Request.	DsUzMediaCostB
S-DSS-00430	The SDSRV CI shall accept the amount of media utilized from the distribution services.	DsUzMediaCostB
S-DSS-00440	The SDSRV CI shall be capable of providing estimated Service Request Cost.	DsClRequest

Table A-1. DSS Release B Requirements Trace (3 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00730	The SDSRV CI shall provide the capability to store Metadata problem reports.	DsNsMPR
S-DSS-00732	The SDSRV CI shall provide the capability for one Data Server to accept Data Availability Schedules from another Data Server.	DsClRequest
S-DSS-00740	The SDSRV CI shall notify operations staff of the receipt of Metadata problem reports.	MSS
S-DSS-00750	The SDSRV CI shall provide Metadata problem reports to operations staff upon request.	DsNsMPR
S-DSS-00760	The SDSRV CI shall provide application program interfaces to all the operator functions.	Admin CSC
S-DSS-00770	The SDSRV CI shall utilize vendor supplied tools to analyze system CPU performance.	MSS
S-DSS-00780	The SDSRV CI shall utilize vendor supplied tools to monitor the performance of query processing.	Illustra, if applicable
S-DSS-00790	The SDSRV CI shall utilize vendor supplied tools to analyze system storage performance.	MSS
S-DSS-00800	The SDSRV CI shall utilize vendor supplied tools to tune system throughput performance.	MSS
S-DSS-00810	The SDSRV CI shall utilize vendor supplied tools to analyze system throughput performance.	MSS
S-DSS-00822	The SDSRV CI shall collect and provide Configuration Management data to the MSS using a MSS provided Configuration Management API.	EcAgManager
S-DSS-00823	The SDSRV CI shall collect and provide Accounting Management data to the MSS using a MSS provided Accounting Management API.	EcRequest, EcAgManager
S-DSS-00824	The SDSRV CI shall collect and provide Accountability Management data to the MSS using a MSS provided Accountability Management API.	EcAcOrderCMgr, EcAgManager
S-DSS-00825	The SDSRV CI shall collect and provide Performance Management data to the MSS using a MSS provided Performance Management API.	EcAgManager
S-DSS-00826	The SDSRV CI shall collect and provide Security Management data to the MSS using a MSS provided Security Management API.	EcAgManager
S-DSS-00827	The SDSRV CI shall collect and provide Scheduling Management data to the MSS using a MSS provided Scheduling Management API.	EcAgManager
S-DSS-00830	The SDSRV CI shall collect Fault Management Data, such as, device failures, Service Request failures, transmission failures and general failures. This information shall be sent to the SMC for fault isolation.	EcAgManager
S-DSS-00840	The SDSRV CI shall inform the collocated eleements of ECS if resource availability falls below nominal operating parameters. This applies to staging resources and peripheral resources.	EcAgManager

Table A-1. DSS Release B Requirements Trace (4 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00920	The SDSRV CI shall provide Logistics Status to the SMC.	EcPfManagedServer
S-DSS-00930	The SDSRV CI shall provide training information to the SMC.	EcPfManagedServer
S-DSS-00980	The SDSRV CI operations staff shall have the capability to receive from the SMC, maintenance directives.	EcPfManagedServer
S-DSS-00990	The SDSRV CI operations staff shall have the capability to receive from the SMC, directives for integration, testing, and simulation.	EcPfManagedServer
S-DSS-01000	The SDSRV CI operations staff shall have the capability to receive from the SMC, configuration management directives.	EcPfManagedServer
S-DSS-01010	The SDSRV CI operations staff shall have the capability to receive from the SMC, logistics management directives.	EcPfManagedServer
S-DSS-01020	The SDSRV CI operations staff shall have the capability to receive from the SMC fault management directives.	EcPfManagedServer
S-DSS-01030	The SDSRV CI operations staff shall have the capability to receive from the SMC security directives.	EcPfManagedServer
S-DSS-01035	The SDSRV CI operations staff shall have the capability to receive from the SMC scheduling directives, and scheduling adjudication directives.	EcPfManagedServer
S-DSS-01040	The SDSRV CI operations staff shall provide integration, testing, and simulation status to the SMC.	EcPfManagedServer
S-DSS-01050	The SDSRV CI operations staff shall have the capability to receive training management directives from the SMC.	EcPfManagedServer
S-DSS-01080	The SDSRV CI shall notify operations staff in the event that data required for an on-demand data production is not accessible.	MSS
S-DSS-01170	The SDSRV CI shall provide the capability to monitor resource utilization on a client basis.	DsUzResourceCostB
S-DSS-01200	The SDSRV CI shall notify the requester in the event that an on-demand data production cannot be completed.	DsClRequest
S-DSS-01220	The SDSRV CI shall provide the capability for a client to suspend processing of a client session.	DsClESDTReferenceCollector
S-DSS-01290	The SDSRV CI shall provide the capability for the operations staff to suspend all active client sessions.	DsAdRequestIF
S-DSS-01300	The SDSRV CI shall provide the capability for the operations staff to resume any or all client sessions, previously suspended by operations staff or clients.	DsAdRequestIF
S-DSS-01310	The SDSRV CI shall provide the capability for the client to resume a client session, previously suspended by the client.	DsClESDTReferenceCollector
S-DSS-01320	The SDSRV CI shall provide the capability for the operations staff to terminate any or all active or suspended client sessions.	DsAdRequestF

Table A-1. DSS Release B Requirements Trace (5 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-01330	The SDSRV CI shall provide the capability for the client to terminate any or all active or suspended client sessions that were previously initiated by the client.	DsClESDTReferenceCollector
S-DSS-01360	The SDSRV CI shall, in the event of a restart after a processing failure, recover the state of all Service Requests, including the rollback of all incomplete Data Base Transactions, and the recovery of all complete Data Base Transactions.	DsSrRequestVector, DsSrCatalog
S-DSS-01410	The SDSRV CI shall log the suspension of the processing of a Service Request or the suspension of a client session.	DsSrConnection
S-DSS-01420	The SDSRV CI shall log the resumption of a previously suspended Service Request or client session.	DsSrConnection
S-DSS-01440	The SDSRV CI shall provide client Session Status Information to the requester.	DsClESDTReferenceCollector
S-DSS-01450	The SDSRV CI shall provide application programming interfaces capable of supporting the development of extensions for the addition of Metadata fields that are unique to the data maintained at a specific DAAC.	DsDeDescriptor
S-DSS-01474	The SDSRV CI shall validate Subscription Requests for time interval events. Time intervals will be limited to daily, weekly, or monthly.	IDG
S-DSS-01520	The SDSRV CI shall provide the capability to notify a user that a new version of the data has been archived.	DsClSubscription
S-DSS-01540	The SDSRV CI shall provide the capability to bundle notification of discrete events into a single notice to the subscriber.	IDG
S-DSS-01560	The SDSRV CI shall accept Subscription Update Requests to update stored Subscriptions by changing the event or the action.	IDG
S-DSS-01580	The SDSRV CI shall provide the capability for operations staff to update the stored Subscriptions by changing the event and/or action.	IDG
S-DSS-01590	The SDSRV CI shall provide the capability for a user client to update their stored Subscriptions by changing the action and/or event.	IDG
S-DSS-01620	The SDSRV CI shall validate that Subscription Update Requests specify a valid Subscription Identifier and a valid replacement Subscription.	IDG
S-DSS-01700	The SDSRV CI shall periodically report on new events for timer-based Subscriptions and will not repeat notification of old events.	IDG
S-DSS-01790	The SDSRV CI shall provide access to compound data type services.	DsClRequest
S-DSS-02901	The SDSRV CI shall provide the capability to subset, subsample, or average data within a granule based on Geographic location for products specified in Appendix F - Data Type Matrix.	DsGeECSDataProduct

Table A-1. DSS Release B Requirements Trace (6 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-02902	The SDSRV CI shall provide the capability to subset, subsample, or average data within a granule based on Spectral band for products specified in Appendix F - Data Type Matrix.	DsGeECSDataProduct
S-DSS-02903	The SDSRV CI shall provide the capability to subset, subsample, or average data within a granule based on Time for products specified in Appendix F - Data Type Matrix.	DsGeECSDataProduct
S-DSS-02904	The SDSRV CI shall provide the capability to subset, subsample, or average data within a granule based on WRS for products specified in Appendix F - Data Type Matrix.	DsGeECSDataProduct
S-DSS-03002	The SDSRV CI shall be capable of receiving L0 - L4 Data.	DsGeESDT
S-DSS-03004	The SDSRV CI shall be capable of receiving Ancillary Data.	DsNpNonECSDataProduct
S-DSS-03006	The SDSRV CI shall be capable of receiving Metadata associated with Ancillary Data.	DsNpNonECSDataProduct
S-DSS-03050	The SDSRV CI shall be capable of receiving FDF Orbit Data for AM1 Instruments.	DsGeESDT
S-DSS-03060	The SDSRV CI shall be capable of receiving FDF Attitude Data for AM1 Instruments.	To be Deleted
S-DSS-03100	The SDSRV CI shall be capable of receiving FDF Metadata for Orbit and Attitude data for AM1 Instruments.	DsGeESDT
S-DSS-03122	The SDSRV CI shall be capable of receiving real EOS instrument data to support pre-launch checkout of the ground system.	DsGeESDT
S-DSS-03124	The SDSRV CI shall be capable of receiving simulated EOS instrument data to support pre-launch checkout of the ground system.	DsNpNonECSDataProduct
S-DSS-03190	The SDSRV CI shall be capable of receiving Orbit/Attitude data.	DsGeESDT
S-DSS-03200	The SDSRV CI shall be capable of receiving Metadata associated with Orbit/Attitude data.	DsGeESDT
S-DSS-03290	The SDSRV CI shall be capable of receiving Spacecraft Historical Data.	DsGeESDT
S-DSS-03330	The SDSRV CI shall be capable of receiving TBD Special Data Products.	DsNpNonECSDataProduct
S-DSS-03340	The SDSRV CI shall be capable of receiving Metadata associated with TBD Special Data Products.	DsNpNonECSDataProduct
S-DSS-03361	The SDSRV CI shall be capable of receiving NMC data.	DsNpNonECSDataProduct
S-DSS-03362	The SDSRV CI shall be capable of receiving First Look Products from the DAO.	DsGeESDT
S-DSS-03363	The SDSRV CI shall be capable of receiving Reanalysis Products from the DAO.	DsNpNonECSDataProduct
S-DSS-03364	The SDSRV CI shall be capable of receiving Final Analysis Products from the DAO.	DsGeESDT

Table A-1. DSS Release B Requirements Trace (7 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-03400	The SDSRV CI shall verify compliance of scientist provided data with EOSDIS defined standards for file content and structure (not scientific content).	Ingest
S-DSS-03410	The SDSRV CI shall verify compliance of scientist provided Metadata with EOSDIS defined standards for Metadata content and structure (not scientific content).	DsDeDescriptor
S-DSS-03460	The SDSRV CI shall interface with the STMGT CI to provide storage for FDF Orbit Data for AM-1 instruments.	DsGeESDT
S-DSS-03470	The SDSRV CI's MD Component shall provide storage for Metadata associated with FDF Orbit and Attitude Data for AM-1 instruments.	DsMdCatalog
S-DSS-03492	The SDSRV CI shall interface with the STMGT CI to provide storage for real EOS instrument data to support pre-launch checkout of the ground system.	DsGeESDT
S-DSS-03494	The SDSRV CI shall interface with the STMGT CI to provide storage for simulated EOS instrument data to support pre-launch checkout of the ground system.	DsGeESDT
S-DSS-03600	The SDSRV CI shall interface with the STMGT CI to provide storage for production plan data.	DsGeESDT
S-DSS-03660	The SDSRV CI shall interface with the STMGT CI to provide storage for spacecraft historical data.	DsGeESDT
S-DSS-03700	The SDSRV CI shall interface with the STMGT CI to provide storage for TBD special Data Products.	DsGeESDT
S-DSS-03710	The SDSRV CI's MD Component shall provide storage for Metadata associated with TBD special Data Products.	DsMdCatalog
S-DSS-03741	The SDSRV CI shall interface with the STMGT CI to provide storage for NMC data.	DsGeESDT
S-DSS-03742	The SDSRV CI shall interface with the STMGT CI to provide storage for First Look Products.	DsGeESDT
S-DSS-03743	The SDSRV CI shall interface with the STMGT CI to provide storage for Reanalysis Products.	DsGeESDT
S-DSS-03744	The SDSRV CI shall interface with the STMGT CI to provide storage for Final Analysis Products.	DsGeESDT
S-DSS-03940	The SDSRV CI shall be capable of receiving estimated disk utilization from the PLANG CI.	DsSrCost
S-DSS-03950	The SDSRV CI shall be capable of receiving estimated CPU utilization from the PLANG CI.	DsSrCost
S-DSS-03960	The SDSRV CI shall be capable of receiving estimated disk utilization from the STMGT CI.	DsSrCost
S-DSS-03990	The SDSRV CI shall be capable of receiving actual disk utilization from the PLANG CI.	DsSrCost
S-DSS-03992	The SDSRV CI shall interface with the STMGT CI to provide storage for real EOS instrument data to support pre-launch checkout of the ground system.	DsGeESDT

Table A-1. DSS Release B Requirements Trace (8 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-03994	The SDSRV CI shall interface with the STMGT CI to provide storage for simulated EOS instrument data to support pre-launch checkout of the ground system.	DsGeESDT
S-DSS-04000	The SDSRV CI shall be capable of receiving actual CPU utilization from the PLANG CI.	DsSrCost
S-DSS-04010	The SDSRV CI shall be capable of receiving actual disk utilization from the STMGT CI.	DsSrCost
S-DSS-04038	The SDSRV CI shall supply L0 - L4 Data to the DDIST CI.	DsGeESDT
S-DSS-04080	The SDSRV CI shall supply FDF orbit data for AM-1 instruments packages to the DDIST CI.	DsGeESDT
S-DSS-04082	The SDSRV CI shall supply FDF attitude data for AM-1 instruments packages to the DDIST CI.	DsGeESDT
S-DSS-04112	The SDSRV CI shall be capable of supplying real EOS instrument data to support pre-launch checkout of the ground system to the DDIST CI.	DsGeESDT
S-DSS-04114	The SDSRV CI shall be capable of supplying simulated EOS instrument data to support pre-launch checkout of the ground system to the DDIST CI.	DsGeESDT
S-DSS-04180	The SDSRV CI shall supply Orbit/Attitude Data to the DDIST CI.	DsGeESDT
S-DSS-04190	The SDSRV CI's MD Component shall supply Metadata associated with Orbit/Attitude Data to the DDIST CI.	DsMdCatalog
S-DSS-04320	The SDSRV CI shall supply TBD special Data Products to the DDIST CI.	DsGeESDT
S-DSS-04330	The SDSRV CI's MD Component shall supply Metadata associated with TBD special Data Products to the DDIST CI.	DsMdCatalog
S-DSS-04332	The SDSRV CI shall supply Research results (articles, algorithms, data sets, software) to the DDIST CI.	DsGeESDT
S-DSS-04340	The SDSRV CI shall supply V0 migration Data Products to the DDIST CI.	DsGeESDT
S-DSS-04350	The SDSRV CI shall supply Metadata associated with V0 migration Data Products to the DDIST CI.	DsGeESDT
S-DSS-04351	The SDSRV CI shall supply NMC data to the DDIST CI.	DsGeESDT
S-DSS-04352	The SDSRV CI shall supply First Look Products to the DDIST CI.	DsGeESDT
S-DSS-04353	The SDSRV CI shall supply Reanalysis Products to the DDIST CI.	DsGeESDT
S-DSS-04354	The SDSRV CI shall supply Final Analysis Products to the DDIST CI.	DsGeESDT
S-DSS-04410	The SDSRV CI's MD Component shall have the ability to store references to Orbit/Attitude Data as Metadata for science data.	DsMdCatalog
S-DSS-04500	The SDSRV CI's MD Component shall have the ability to indicate the need for on-demand product generation as Metadata for science data.	DsMdCatalog

Table A-1. DSS Release B Requirements Trace (9 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-04620	The SDSRV CI shall update the Metadata for a data item that has been purged from the system.	DsMdCatalog
S-DSS-04630	The SDSRV CI shall update the Metadata whenever a data item is relocated to another site.	DsMdCatalog
S-DSS-04695	The SDSRV CI shall provide the capability to perform searches that identify the source instrument for a specified data product.	DsMdCatalog
S-DSS-04720	The SDSRV CI shall provide DARs to ASTER ICC.	Client Subsystem
S-DSS-04730	The SDSRV CI shall accept DARs from the client.	Client Subsystem
S-DSS-04740	The SDSRV CI shall provide DAR status to the client, in response to DAR Status Requests.	Client Subsystem
S-DSS-04745	The SDSRV CI shall provide operations staff with the ability to display and list outstanding DARs that are accessible by the Data Server.	Client Subsystem
S-DSS-04750	The SDSRV CI shall accept DAR Status from IPs	Client Subsystem
S-DSS-04760	The SDSRV CI shall accept Subscription Requests from the client linked to a specified, existing DAR.	IDG
S-DSS-04770	The SDSRV CI shall send DAR Status Requests to ASTER ICC.	Client Subsystem
S-DSS-04780	The SDSRV CI shall receive DAR Status from the ASTER ICC.	Client Subsystem
S-DSS-04800	SDSRV CI mode-specific applications shall access data only for the mode in which the application is configured.	EcPfGenProcess
S-DSS-04810	SDSRV CI mode-specific applications shall be capable of simultaneous execution in different modes on the same machine.	EcPfGenProcess
S-DSS-04820	SDSRV CI mode-specific applications shall be capable of simultaneous execution in different modes on different machines.	EcPfGenProcess
S-DSS-04830	The SDSRV CI shall be capable of using simulated time values supplied by CSS, when executing in a non-production mode.	EcPfGenProcess, EcUtTime
S-DSS-04840	SDSRV CI server applications shall register within their mode-associated namespace in the CSS name service.	EcPfConfigFile
S-DSS-04850	SDSRV CI client applications shall incorporate a mode identifier for CSS name service lookups.	EcPfConfigFile
S-DSS-04860	SDSRV CI mode-specific executables and scripts shall accept a specific mode only at startup.	EcPfConfigFile
S-DSS-04870	The SDSRV CI shall include the mode identifier in activity log record entries for cost and accounting data.	DsUzResourceCostB
S-DSS-21280	The SDSRV CI shall provide application programming interfaces (APIs) to support Insert Requests.	DsClRequest
S-DSS-00851	The DDSRV CI shall collect and provide Configuration Management data to the MSS using a MSS provided Configuration Management API.	EcAgManager

Table A-1. DSS Release B Requirements Trace (10 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-00852	The DDSRV CI shall collect and provide Accounting Management data to the MSS using a MSS provided Accounting Management API.	EcRequest, EcAgManager
S-DSS-00853	The DDSRV CI shall collect and provide Accountability Management data to the MSS using a MSS provided Accountability Management API.	EcAcOrderCMgr, EcAgManager
S-DSS-00854	The DDSRV CI shall collect and provide Performance Management data to the MSS using a MSS provided Performance Management API.	EcAgManager
S-DSS-00855	The DDSRV CI shall collect and provide Security Management data to the MSS using a MSS provided Security Management API.	EcAgManager
S-DSS-00856	The DDSRV CI shall collect and provide Scheduling Management data to the MSS using a MSS provided Scheduling Management API.	EcAgManager
S-DSS-10020	The DDSRV CI shall accept Subscriptions for metadata from the client.	To Be Deleted
S-DSS-10055	The DDSRV CI shall provide, to qualified users, access to all documents and data types held in the server's collection.	DsSeWWWServer; DsCtSearchCommand; DsCtAcquireCommand
S-DSS-10202	The DDSRV CI shall provide the capability to ingest documentation in Microsoft WORD format.	DsCdCSDT; DsCdTypeID; DsCdWordB
S-DSS-10206	The DDSRV CI shall provide the capability to ingest documentation in Interleaf format.	DsCdCSDT; DsCdTypeID; DsCdInterleafB
S-DSS-10208	The DDSRV CI shall provide the capability to ingest documentation in WordPerfect format.	DsCdCSDT; DsCdTypeID; DsCdWordPerfectB
S-DSS-10230	The DDSRV CI shall provide application programming interfaces that support addition of documents for use as Guide data for DAAC-specific Data Products.	DDSRV CI
S-DSS-10231	The DDSRV CI shall utilize vendor supplied tools to analyze system CPU performance.	MSS Subsystem
S-DSS-10232	The DDSRV CI shall utilize vendor supplied tools to analyze system throughput performance.	MSS Subsystem
S-DSS-10233	The DDSRV CI shall collect Fault Management Data, such as, device failures, Service Request failures, transmission failures and general failures. This information shall be sent to the SDSRV CI for forwarding to the SMC for fault isolation.	MSS Subsystem
S-DSS-10260	The DDSRV CI shall provide application programming interfaces that support development of extensions for addition of documents for use as Guide data for DAAC-specific Data Products.	DDSRV CI
S-DSS-10300	The DDSRV CI shall complete a search for a guide document by a single keyword in not exceeding 8 seconds.	DDSRV CI
S-DSS-10305	The DDSRV CI shall complete a directory search using a single keyword in a period to exceed 8 seconds.	DDSRV CI

Table A-1. DSS Release B Requirements Trace (11 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-10306	The DDSRV CI shall complete a directory search using multiple keywords in a period not to exceed 13 seconds.	DDSRV CI
S-DSS-10310	The DDSRV CI shall complete a keyword search on a 1000 page document of not exceeding 3 seconds.	DDSRV CI
S-DSS-10400	DDSRV CI mode-specific applications shall access data only for the mode in which the application is configured.	EcPfGenProcess
S-DSS-10410	The DDSRV CI mode-specific applications shall be capable of simultaneous execution in different modes on the same machine.	EcPfGenProcess
S-DSS-10420	DDSRV CI mode-specific applications shall be capable of simultaneous execution in different modes on different machines.	EcPfGenProcess
S-DSS-10430	The DDSRV CI shall be capable of using simulated time values, supplied by CSS, when executing in a non-production mode.	EcPfGenProcess; EcUtTime
S-DSS-10440	DDSRV CI server applications shall register within their mode-associated namespace in the CSS name service.	EcPfConfigFile
S-DSS-10450	DDSRV CI client applications shall incorporate a mode identifier for CSS name service lookups.	EcPfConfigFile
S-DSS-10460	DDSRV CI mode-specific executables and scripts shall accept a specific mode only at startup.	EcPfConfigFile
S-DSS-10465	The DDSRV CI shall provide the capability to store and retrieve test plans and test procedures.	DsCtSearchCommand, DsCtAquireCommand, DsCtInsertCommand
S-DSS-10470	DDSRV CI shall include the mode identifier in activity log record entries for cost and accounting data.	DsUzResourceCostB
S-DSS-20162	The STMGT CI shall provide the capability to mount on-line backup media via automated means.	DsStArchive
S-DSS-20171	The STGMT CI shall provide operations personnel with the capability to screen the archive holdings for lost volumes.	DsStArchive
S-DSS-20210	For any EOS Level 0 or L1A (if L0 is not available) data item that can not be located or is inaccessible and can not be re-created, the STMGT CI shall notify the operator which data item is missing and the operator shall request the data item be re-ingested from EDOS.	DsStArchive
S-DSS-20260	For each piece of archive media, the STMGT CI shall provide the capability to display the length of time to store data on the media before deletion.	DsStArchive
S-DSS-20270	The STMGT CI shall provide the capability to change the length of time to store data on archive media before deletion of the data.	DsStArchive
S-DSS-20280	The STMGT CI shall provide the capability to directly notify active users when Data Products will be deleted.	SDSRV
S-DSS-20290	The STMGT CI shall provide the capability to indirectly notify users when Data Products will be deleted via a bulletin board type mechanism.	SDSRV

Table A-1. DSS Release B Requirements Trace (12 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-20450	The STMGT CI shall provide the capability to archive real EOS instrument data to support pre-launch checkout of the ground system.	DsStArchive
S-DSS-20455	The STMGT CI shall provide the capability to retrieve real EOS instrument data to support pre-launch check out of ground systems.	DsStArchive
S-DSS-20457	The SDSRV CI shall interface with the STMGT CI to provide storage for real EOS instrument data to support pre-launch instrument checkout.	DsStArchive
S-DSS-20460	The STMGT CI shall provide the capability to archive simulated EOS instrument data to support pre-launch checkout of the ground system.	DsStArchive
S-DSS-20462	The STMGT CI shall provide the capability to retrieve simulated EOS instrument data to support pre-launch checkout of the ground system.	DsStArchive
S-DSS-20465	The SDSRV CI shall interface with the STMGT CI to provide storage for simulated EOS instrument data to support pre-launch instrument checkout.	DsStArchive
S-DSS-20470	The STMGT CI shall provide the capability to retrieve simulated EOS instrument data to support pre-launch checkout of the ground system.	DsStArchive
S-DSS-20550	The STMGT CI shall provide operations staff a mechanism to display/view storage system operating parameters which affect storage system performance.	DsStStagingMonitor, DsStPullMonitor
S-DSS-20560	The STMGT CI shall provide operations staff a mechanism to display/view storage system operating parameters which affect storage system scheduling.	DsStResourcePolicy
S-DSS-20570	The STMGT CI shall provide operations staff the capability to change storage system operating parameters which affect storage system performance.	DsStStagingMonitor, DsStPullMonitor
S-DSS-20580	The STMGT CI shall provide operations staff the capability to change storage system operating parameters which affect storage system scheduling.	DsStResourcePolicy
S-DSS-20610	The STMGT CI shall provide the capability to archive multiple versions of Data Granules.	DsStArchive
S-DSS-20624	The STMGT CI shall provide a mechanism to statistically monitor the checksum error rate of archive media.	DsStArchive
S-DSS-20625	The STMGT CI shall allow the operator to manually specify archive media to be recopied/refreshed.	FSMS COTS
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-20720	The STMGT CI shall provide a mechanism to mark data for deletion. The mechanism shall be based on selection of max time to store data before it's deleted from storage. It shall also mark earlier versions when multiple versions have been archived.	DsStArchive

Table A-1. DSS Release B Requirements Trace (13 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-20730	The STMGT CI shall provide a mechanism to automatically delete archived data which has been marked for deletion.	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-20750	For data retrieval requests for L0 data from EDOS, STMGT CI shall satisfy such requests with appropriate L0 or L1A data. Note: These instruments provide L0 data, CERES, LIS, ASTER, MISR, MODIS, MOPPIT; these provide L1A data, LIS, PR, TMI, VIRS.	DsStArchive
S-DSS-20800	The STMGT CI shall use operator selectable criteria to determine the physical storage device that data types will be stored in. This criteria shall consider: current store and retrieval activity, number of storage devices, type of data to be stored.	DsStResourceConfig
S-DSS-20810	The STMGT CI shall provide operations staff the capability to manually alter the criteria that determines the physical storage device that data sets will be stored in.	DsStResourceConfig
S-DSS-20820	The STMGT CI shall provide operations staff the capability to alter the criteria that determines removal of archive media from storage devices to allow insertion of new or different archive media in the storage device.	DsStArchive
S-DSS-20830	In determining the archive media to be removed the criteria shall consider the media's capacity for storing additional data, the last time data was accessed on the media and whether the media is currently in use to store or retrieve data.	DsStArchive
S-DSS-20840	The STMGT CI shall report information on the storage system. Information reported shall include file access time, file accesses per hour, size of files stored onto archive media, size of files retrieved from archive media, amount of storage allocated.	DsStArchive, DsStStagingDisk
S-DSS-20850	The STMGT CI shall collect information on the storage system. Information shall include avg access time, avg # of accesses per hour, mean request inter-arrival time, avg file size stored, avg file size retrieved and avg file residency time on disk.	DsStArchive DsStStagingDisk
S-DSS-20860	The STMGT CI shall provide a mechanism to monitor the performance of the ECS archival storage system.	DsStResourceManager
S-DSS-20870	The STMGT CI shall provide operations staff the capability to view/display performance information on the storage system.	DsStResourceManager
S-DSS-21130	The STMGT CI shall provide estimates of staging device time delays for subsetted Data Requests.	DsStRequestManager
S-DSS-21140	The STMGT CI shall provide estimates of staging device time delays for subsampled Data Requests.	DsStRequestManager
S-DSS-21150	The STMGT CI shall provide estimates of staging device time delays for summary Data Requests.	DsStRequestManager

Table A-1. DSS Release B Requirements Trace (14 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-21240	The STMGT CI shall provide operations staff a mechanism to display/view storage system utilization by ECS element.	DsStStagingDisk
S-DSS-21250	The STMGT CI shall provide operations staff a mechanism to display/view storage system performance by ECS element.	DsStResourceManager
S-DSS-21260	The STMGT CI shall provide operations staff a mechanism to display/view storage system cost by ECS element.	DsStResourceManager
S-DSS-21290	The STMGT CI shall provide application programming interfaces (APIs) to support Retrieval Requests.	DsStArchive
S-DSS-21300	The STMGT CI shall provide application programming interfaces (APIs) to support Status Requests related to previous Insert Requests.	DsStRequestManager
S-DSS-21310	The STMGT CI shall provide application programming interfaces (APIs) to support Status Requests related to previous Retrieval Requests.	DsStRequestManager
S-DSS-21320	The STMGT CI shall provide the capability to estimate time delays for data retrievals due to contention for hardware resources.	DsStRequestManager
S-DSS-21340	The STMGT CI shall provide data to support administrative requests for Accounting Management Data.	DsStResourceSchedule
S-DSS-21350	The STMGT CI shall collect Accounting Management Data as defined in Appendix A.	DsStResourceSchedule
S-DSS-21430	The STMGT CI shall provide operations staff a mechanism to delete records from the File Directory.	DsStArchive
S-DSS-21610	The MSFC DAAC Science Management within the Data Server shall make TSDIS original standard products (Level 1B-3) eligible for deletion after 6 months	DSS Subsystem
S-DSS-30190	The DDIST CI shall record the cost of the shipping and handling of the media associated with each Media Distribution request.	DsDdMedia
S-DSS-30200	The DDIST CI shall record the network cost of data transmission, the User Identifier and the Request Identifier.	DsDdMedia
S-DSS-30210	The DDIST CI shall record the cost of CPU intensive operations performed on data to be distributed. Such operations include compression/decompression and reformatting.	DsDdMedia
S-DSS-30220	The DDIST CI shall record the cost of archive storage for data to be distributed based on distribution size.	DsDdMedia
S-DSS-30230	The DDIST CI shall provide the capability to report the estimated media utilization to the SDSRV CI.	DsDdDistList
S-DSS-30240	The DDIST CI shall provide the capability to report the actual media utilization to the SDSRV CI.	DsDdDistList
S-DSS-30245	The DDIST CI shall provide the capability to report accounting data to the SDSRV CI.	DsDdDistList

Table A-1. DSS Release B Requirements Trace (15 of 15)

L4 Rqmt ID	L4 Requirement Text	Subsystem, Object Class, CSC, or CI
S-DSS-30296	The DDIST CI shall alert SMC when electronic transmission problems are encountered.	DsDdPushMedia
S-DSS-30450	The DDIST CI shall provide the capability to distribute on 4mm tape.	DsDdTapeMedia
S-DSS-30460	The DDIST CI shall provide the capability to distribute on 3480/3490 tape.	DsDdTapeMedia
S-DSS-30482	The DDIST CI shall provide the capability to support additional data distribution formats and conversion software.	DsDdDistList
S-DSS-30500	If the number of correctable errors exceed a system threshold for a piece of media, the DDIST CI shall abort the operation and automatically request a new piece of media from operations staff.	DsDdMedia
S-DSS-30510	Operations staff shall have the capability to specify a threshold of correctable errors for each type of distribution media.	DsDdMedia, DsStResourceConfig
S-DSS-30620	The DDIST CI shall provide the capability to distribute documents electronically via FAX transmissions.	DsDdFaxMediaB
S-DSS-30690	For physical media distributions, the DDIST CI shall generate a physical "media label" that operations staff can apply to the media, and shall associate the individual piece of media with any other media in the distribution.	DsDdMediaLabelB
S-DSS-30700	For physical media distributions, the DDIST CI shall generate a physical "shipping label" that operations staff can affix to the shipping container and indicates the destination of the media.	DsDdShippingLabelB
S-DSS-30770	The DDIST CI shall provide an applications program interface to submit Distribution Requests, obtain Request Status for Distribution Requests, and retrieve a list of Distribution Requests submitted.	DsDdDistList, DsDdDistRequest, DsDdRequestList
S-DSS-30795	For physical media distributions, the DDIST CI shall record the cost of the media to be used for accounting.	DsDdMedia

This page intentionally left blank.