GSFC JPSS CMO
November 11, 2013
Released

Effective Date: November 7, 2013
Block/Revision: 0123A

Joint Polar Satellite System (JPSS) Ground Project
Code 474
474-00019-01-B0123

Joint Polar Satellite System (JPSS) Application Programming Interface (API) User's Guide, Volume I – C++

For Public Release

The information provided herein does not contain technical data as defined in the International Traffic in Arms Regulations (ITAR) 22 CFC 120.10. This document has been approved For Public Release to the NOAA Comprehensive Large Array-data Stewardship System (CLASS).

Block 1.2.3

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

Joint Polar Satellite System (JPSS)
Application Programming Interface (API)
User's Guide, Volume I – C++

JPSS Electronic Signature Page

Prepared By:

Thomas Jennings
JPSS Ground Project System Engineer
(Electronic Approvals available online at https://jpssmis.gsfc.nasa.gov/mainmenu_dsp.cfm)

Reviewed By:

Leslye Boyce
JPSS Ground Project Systems Manager
(Electronic Approvals available online at https://jpssmis.gsfc.nasa.gov/mainmenu_dsp.cfm)

Goddard Space Flight Center
Greenbelt, Maryland

Preface

This document is under JPSS Ground ERB configuration control. Once this document is approved, JPSS approved changes are handled in accordance with Class I and Class II change control requirements as described in the JPSS Configuration Management Procedures, and changes to this document shall be made by complete revision.

Any questions should be addressed to:

JPSS Configuration Management Office
NASA/GSFC
Code 474
Greenbelt, MD 20771

Change History Log

Revision	Effective Date	Description of Changes (Reference the CCR & CCB/ERB Approval Date; for first Block Version Release, identify origin of document source)	Sections Affected
0123-	Aug 23, 2012	This version incorporates Rev- of 474-00019-01-B0122, dated January 26, 2012, to create the baseline for Block 1.2.3, Rev-. This was approved (out of board) by the JPSS Ground ERB via 474-CCR-12-0512 on the effective date shown.	All
0123-1	Oct 24, 2013	This version incorporates 474-CCR-13-1216 (ECR-CGS-0209) which was approved by the JPSS Ground ERB on the effective date shown	3.0 4.0 Table 4.3.1-1 6.2.5.2.10 6.2.11.3.7 6.2.11.3.14 6.2.11.3.17 6.2.18 Figure 6.2.18-1 6.2.18.1 6.2.18.1.1 6.2.18.1.2 6.2.18.1.3
0123A	Nov 07, 2013	This version incorporates 0123-1 and 474-CCR-13-0998. This was approved by the JPSS Ground ERB on the effective date shown.	All

Northrop Grumman Space & Mission Systems Corp. Space Technology

One Space Park
Redondo Beach, CA 90278

Engineering and Manufacturing Development (EMD)
Acquisition & Operations Contract

CAGE NO. 11982

National Polar-Orbiting Operational Environmental Satellite System (NPOESS) Application Programming Interface (API) User's Guide, Volume I

DOCUMENT DATE: 08/19/2009 **DOCUMENT NO. D41044-01**
CDRL A009 **REV. C**

POINT OF CONTACT: Terri Matthews, Systems Integration

ELECTRONIC APPROVAL SIGNATURES:

Clark Snodgrass, SEITO Director

Fabrizio Pela, SEIT IPT Lead

Keith Reinke, Ground Segments IPT Lead

Mary Ann Chory, Space Segment IPT Lead

Ben James, Operations and Support IPT Lead

David Vandervoet, NPOESS Program Manager

Prepared by
Northrop Grumman Space Technology
One Space Park
Redondo Beach, CA 90278

Under
Contract No. F04701-02-C-0502

Prepared for
Department of the Air Force
NPOESS Integrated Program Office
C/O SMC/CIK
2420 Vela Way, Suite 1467-A8
Los Angeles AFB, CA 90245-4659

Northrop Grumman Space & Mission Systems Corp.
Space Technology
One Space Park
Redondo Beach, CA 90278

NORTHROP GRUMMAN
Raytheon

Revision/Change Record

For Document No.
D41044-01

Revision	Document Date	Revision/Change Description	Pages Affected
---	03/09/2007	Initial release. Divided into two volumes from the basic document with reorganization of content. Incorporated ECR 560	All
A	06/20/2008	Full revision of the document to incorporate IDPS B1.5 API documentation. This revision of the document changes the content and structure of the API for the C++, Java, and JMS APIs The Revision, ECR 778, also incorporates the following DCOs: DCO A1 D41044-01 NPOESS API User's Guide Vol.I ECR 654 – C API Effectivity ECR 778A removes fillInXML, fillInData, readFile per comments submitted against the ECR. Also, removed spaces from DDDS_PORT and DDDS_HOST definitions.	All
B	01/23/09	ECR 891A is a revision of the document to incorporate comments from CCR and CDA reviews. • Added the NPOESS Software Standard and Practices Manual (SSPM) to the list of reference documents. • Clarified the use of IET in several places and specified the precision of times in several places • Corrected definitions of Latitude in several places as well as the Latitude Values in several of the examples • Miscelaneous grammar changes for clarity	All
C	8/19/09	ECR 985A updated the version number for StorNext ECR 985 is a revision of the document to bring document into alignment with IDPS Build 1.5.X.1. It also updates the versions of compilers and COTS used for the API. • Also incorporates ECR 954, Deletion of C API	All

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

The National Polar-orbiting Operational Environmental Satellite System (NPOESS) Application Programming Interface (API) User's Guide describes how users of NPOESS environmental data can request and receive available data from the NPOESS Interface Data Processing Segment (IDPS) through the IDPS Data Delivery Subsystem (DDS). Users familiar with the GUI interface will notice differences between the GUI and this document. This document does not describe the GUI interface. Instead it is a description of a software to software interface. The GUI interface was built on top of the software in order to make some things (such as time designations) easier for a human to work with. This extra processing is not required by the code, and so there are deltas between this document and the GUI options.

Included within this guide are code signatures which provide guidance for interacting with IDPS/DDS in order to submit requests, customize requests for a variety of unique request types, and specify valid data delivery destinations.

1.1 Scope

This document is intended for any programmer who uses the NPOESS API. This document contains information about the NPOESS API, its coding conventions, and code signatures. It includes the general descriptions of functions, descriptions of request parameters and the effect they have on requests, and descriptions of the various types of requests.

1.2 Volume Overview

Due to the size of the NPOESS API User's Guide, this document is divided into two volumes. See the Document Overviews of each of the volumes for a detailed description of their contents. The volumes are organized in the following manner:

NPOESS API User's Guide Volume I: Contains general information for all users of the NPOESS API, as well as the code signatures for the C++ APIs. Sections 1 through 4 of this volume contain general information which is used by both volumes.

NPOESS API User's Guide Volume II: Contains material that is unique to the Java and Java Messaging Services (JMS) APIs including their code signatures.

1.3 Document Overview

The sections in this volume of the document are organized in the following manner:

Section 1 Introduction – This section provides an overview of the NPOESS API system requirements and summarizes the document layout, scope, and configuration management of this document.

Section 2 Applicable Documents – This section identifies applicable compliance and reference documents. It also establishes an order of precedence in the event of conflict between this document and other documents.

Section 3 API Overview – This section provides an overview of the capabilities of the API, as well as a short discussion of the HDF5 file format and a description of data granules.

Section 4 User Functions – This section describes the common and unique parameters required to generate requests for NPOESS Data Products, Ancillary Data, Auxiliary Data, and the Data Catalog. High-level examples are provided for all of the IDPS DDS request types.

Section 5 Deleted

Section 6 C++ API – This section provides the information for the C++ version of the API.

Appendix A System Requirements –This section contains information pertaining to the software needs of the NPOESS API.

Appendix B Document Specific Acronyms List – Provides a list of acronyms unique to this document. All other acronyms are identified and listed in the NPOESS Acronyms, D35838.

1.3.1 Description of Code Sub-sections

The sections describing the functions available, including the code, are organized in the same way. This layout is applicable to the code section for the C++, Java, and JMS APIs in both Volumes I and II. All of these sections are divided into the following subsections:

- Coding Conventions – contains the programming convention for the relevant language.
- Module Documentation List – lists the various Modules available via the API
    - Class Name

- Class Attributes – if applicable
- Class Enumerations – if applicable
- Class Constructors
- Class Functions

1.4 Configuration Management

The Government NPOESS IPO Level 1 Configuration Control Board (CCB) is the Configuration Management (CM) authority for external documentation. The Government External ICD stakeholders (e.g., NPOESS IPO, NASA, AFWA, FNMOC, NAVO, NOAA/NESDIS, and NOAA/CLASS) participate in this CCB since any change to a Class 1 document is a Class 1 change (as defined in NPOESS System Spec, SY15-0007). Any subsequent change to external documents after the initial baseline requires a Class 1 Engineering Change Request (ECR) and approval by the Government NPOESS IPO Level 1 CCB.

After approval and release, the Shared Configuration Management Office (CMO) performs the Data Management function and has responsibility for this document. Revisions are issued in the form of a complete document release or change pages, as applicable.

2. APPLICABLE DOCUMENTS

2.1 Compliance and Reference Documents

Compliance documents show conformity in fulfilling official program requirements. Compliance documents, whether Government or non-Government, officially form a part of this document to the extent specified herein.

Reference documents provide additional information that may or may not be used to define an interface or service. In those cases where they are not needed to define an interface or service, they provide supplemental or corollary information, e.g., the NPOESS Acronyms, D35838. In this example, the reference provides the definition of the acronyms, but is not needed to develop an interface or service.

Table 2.1-1, Compliance and Reference Documents identifies those documents referenced throughout this document, specifying whether they are compliance or reference.

Table 2.1-1, Compliance and Reference Documents

Document Number	Document Title	Brief Description	Compliance/Reference
D31405	NPOESS Security Implementation Plan (SIP)	Defines the security policies for NPP and NPOESS	Compliance
SY15-0007	NPOESS System Specification	Defines the NPOESS and NPP system level requirements	Compliance
474-00003	Joint Polar Satellite System (JPSS) to National Oceanic and Atmospheric Administration (NOAA) Interface Control Document (ICD)	Defines the external interfaces between NPOESS and the NESDIS Central and between NPOESS and CLASS	Reference
474-00013	Joint Polar Satellite System (JPSS) to Department of Defense (DoD) Interface Control Document (ICD)	Defines the external interfaces between NPOESS and the DoD Centrals	Reference
474-00016	Joint Polar Satellite System (JPSS) to NPOESS Preparatory Project (NPP) Science Data Segment (SDS) Interface Control Document (ICD)	Defines the external interfaces between NPOESS and SDS	Reference
474-00002	Joint Polar Satellite System (JPSS) Common Interfaces and Services (CIS) Interface Control Document (ICD)	Defines the common interfaces and services accessible to various users for MSD or to access NPP/NPOESS data (e.g. Work Request System)	Reference
474-00001	Joint Polar Satellite System (JPSS) Common Data Format Control Book - External (CDFCB-X)	Describes the data content and format of the data distributed via external and/or inter-segment logical interfaces	Reference
D35836	NPOESS Glossary	Provides brief definitions of NPOESS specialized terms	Reference
D35838	NPOESS Acronyms	Provides a list of NPOESS acronyms and their descriptions	Reference
MN60822-PMO-001	NPOESS Software Standard and Practices Manual (SSPM)	Provides Programming Standards and Practices for Software Engineers	Reference
Apache Software Foundation	http://www.apache.org/	Website for Apache Software Foundation	Reference

Document Number	Document Title	Brief Description	Compliance/Reference
Borland	http://www.borland.com/us/products/middleware/index.html	Website for Borland Software and Middleware	Reference
IBM	http://www-306.ibm.com/software/sw-atoz/index.html	Website for IBM products	Reference
Internet Engineering Task Force (IETF)	http://www.ietf.org/	Website for the Internet Engineer Task Force	Reference
Microsoft	http://www.microsoft.com/windows/default.mspx	Website for Microsoft windows	Reference
World Wide Web Consortium (W3C)	http://www.w3.org	Website for the World Wide Web Consortium	Reference

2.2 Precedence

In the event of a conflict between a compliance document listed in Table 2.1-1, Compliance and Reference Documents and the contents of this document, the NGST SEITO organization, in conjunction with the IPO, shall resolve the conflict. For all Class 2 documents, the SEITO organization shall resolve the conflict. In the event of a conflict between this document and a reference document listed in Table 2.1-1, Compliance and Reference Documents, this document takes precedence.

3. API OVERVIEW

The purpose of this document is to provide guidance to users who are writing code or scripts that will request data from the NPOESS IDPS/DDS. This document provides detailed descriptions for the following operations:

- logging on to the system to gain access to the functions and logging off the system
- data request generation and modification
- specification of the data delivery location
- data request submission

To submit a request, the user's software must first log on with a username, password, and user role. The software submits the request supplying a unique Request ID along with the request parameters. Once a system generated Request ID is received in response, the software may perform other functions, request status or request retrieval of templates, destinations, or other data, or it may log itself out of the system. Data is not sent through this interface, so it does not matter whether the software remains logged in to the system or not.

Data request generation requires the user's software to build a request which specifies the desired type of data and the parameters that describe the exact information required. Requests may be built from scratch, or built from templates. IDPS/DDS data includes Raw Data Records (RDRs), Temperature Data Records (TDRs), Sensor Data Records (SDRs), Environmental Data Records (EDRs), Application Related Products (ARPs), certain Intermediate Products (IPs), Auxiliary Data, and Ancillary Data. A user may also request a granule from the data catalog. Only data that is delivered through the IDPS/DDS can be requested. Some data, including items not listed here, such as reports, may be available through other mechanisms. Each request, whether submitted via a GUI or the API, is associated with a user. The user's role may preclude them from receiving certain items, such as some IPs or unofficial Auxiliary Data. For more information about roles, see the appropriate Interface Control Document (JPSS to DoD ICD, 474-00013; JPSS to NOAA ICD, 474-00003; JPSS to NPP SDS ICD, 474-00016; or JPSS CIS ICD,

474-00002).

Some request parameters include a timed process delay, request effectivity, aggregation, and delivery of repaired granules. Request effectivity specifies the time that the request is active, usually specified as a start time and duration. Times are in IDPS Epoch Time (IET). For information regarding IET see the Joint Polar Satellite System (JPSS) Common Data Format Control Book - External (CDFCB-X) Volume I - Overview, 474-00001-01. Where appropriate, parameters can also specify a particular time or geolocation for the data.

The request must also include a destination location for the requested data. Delivery locations may be added to, modified, or deleted from the request. Delivery locations must be fully qualified destinations, with the qualifications based upon the protocol used. For example, an ftp destination would be specified as ftp://user:password@host:port/path, while a SAN File System destination would be file://host/path.

Status of requests can be obtained by logging into the API and requesting status. If status through the API is desired, then the login must be maintained with a persistent process.

Some users are given supervisory duties for a set of users. Software connected to the system using these supervisor IDs have functions available that can be used to manage requests (suspend, resume, delete) for users or to transfer ownership of requests from one user to another.

3.1 HDF Files

The requested data and its associated metadata are wrapped and delivered in HDF5 formatted files. If aggregation is not specified for NPOESS Data Products, each granule that meets the request criteria is shipped in a separate HDF5 file. If aggregation is specified, each aggregate of granules that meets the request criteria and contains the minimum number of granules needed to meet or exceed the requested aggregation length will be sent in a single file. The requested aggregation length is limited to a maximum of 104 minutes. An HDF5 file will not contain data that spans more than the requested aggregation length plus the length of one granule. If the request is filled by data that covers a longer temporal range than this, the data will be broken into multiple HDF5

files, with each file containing data that spans no more than this maximum time. For more information on HDF5 files, refer to the Joint Polar Satellite System (JPSS) Common Data Format Control Book - External (CDFCB-X) Volume I - Overview, 474-00001-01.

3.2 Granules

A granule is a segment of data with the size optimally determined to achieve maximum efficiency for an algorithm class. It is a collection of data associated with an integer number of sensor scans, and its definition may vary for each sensor and data product.

A granule is the smallest data product unit delivered by NPOESS. Granules are defined by time; therefore granule data volumes vary according to instrument. If a request is made such that either the start point or the end point of the request is contained within a granule, the whole granule is delivered. This is true whether the request is made temporally or spatially (using geographical coordinates), and is true of any criteria specified in the request (e.g., ending points, boundary lines). See the JPSS CDFCB-X, Vol. I, for more information.

4. USER FUNCTIONS

This section briefly describes the classes of user functions available to the (software) user. In general, the user must log into the system to create or submit a request, update their destination lists, view the catalog, or manage or edit their requests. The user may also inspect the system messages and product status while logged into the system.

4.1 Initial Connection: How to Connect in order to Login

If the proper parameters are passed to the message object created by an implementation of the API, a DDS Request Server is running, and the firewalls are configured correctly then the API will connect to the DDS Server when the message object is created. The API connection will be kept open by heartbeats until the message object is destroyed. Making a connection to the DDS Server is independent of a login. If a valid connection is made then a login to the Server can be requested. After the application calls login on the message object with a valid login, then the application will be logged into the Server.

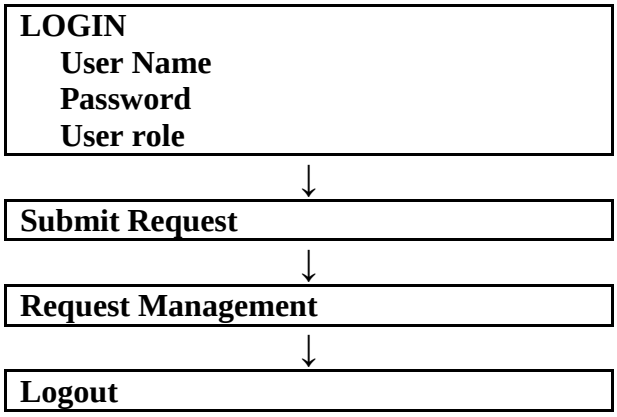
A login will not timeout if a call to an API command, which modifies an API object, is made (add, modify, delete request...etc). If an application does not update an object, then the Server may timeout the application based on the Server's inactivity timeout value.

4.2 Logging into the System

In order to access any of the functions provided by the API, a login into the system must occur. This requires a username, a password, and a user role be provided to the system. The user role determines what data may be accessed. Once the system authenticates and authorizes the login information, requests can be created, submitted, or managed. Requests are processed and the data products are shipped independent of whether the user or the user's software remains logged into the system or not.

Table 4.2-1, Login, Submit Request, and Logout Process, illustrates the steps for logging in, submitting the request, and logging out of the system.

Table 4.2-1, Login, Submit Request, and Logout Process



4.3 Requesting Data Products from the Data Delivery Subsystem (DDS)

This section describes request functions for the NPOESS IDPS/DDS software. Data request generation includes specifying the request type (see Table 4.3.1-1, Request Parameters) and supplying the request criteria, or parameters, required to generate the desired information. Requests consist of several parts, some are common to all requests, and others are unique to a particular request type. The sequence of items in the code is not important, but the user must submit all required parameter information for both common and uniquely-defined parameters. This information can be found in Table 4.3.1-1, Request Parameters. A user can submit requests for any available data allowed by their user role. This section also provides examples for each of the request types, including requests for NPOESS Data Products, Auxiliary Data, and Ancillary Data.

A user can create a request from scratch or use an existing request template in order to create the request. If a request is created from scratch, the user must specify all necessary details about the request, including the request type, destination, and the effectivity (specification of the range of time that the request spans). If the user creates the request from a template, they should modify the Request ID in order to aid traceability and may modify any of the other request parameters. In addition to creating, submitting, and modifying requests, a user may also view, suspend, resume, and delete requests. Request status is kept in the request itself. The user can use the getState function to obtain the request status.

4.3.1 Common User-Supplied Request Parameters

Requests consist of several parts, some common to all requests, and others unique to a particular request type. Common parameters apply to any request, regardless of the request type.

Common required data request parameters include the request name, the product type or category, the data delivery or destination location, and effectivity of the request. Optional common code parameters include process delay, and delivery of repaired data.
Table 4.3.1-1, Request Parameters, defines the parameters that are used in creating requests for data. The request can also be limited in scope by specifying the time or geography with either a temporal or spatial (geographic) subsetting.

The second column of the table specifies the request type which the parameter applies to, and the last column tells you whether or not that parameter is required or optional for the request types the parameter applies to. If the parameter does not apply to a request type, that parameter should not be included in that request.

Table 4.3.1-1, Request Parameters

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description		Required / Optional
Implementation Request Type	N/A	All	Various way Request Types will be implemented. Availability of Request Types is dependent upon role configuration.		Required
			Standard Request	Request products.	
			Temporal Request	Request products temporally.	
			Periodic Request	Requesting Gridded IPs.	
			Catalog Request	Request products from the Data Product Catalog.	
			Catalog Query	Query products from the Data Product Catalog.	

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description		Required / Optional
Request Type	All	N/A	The user must specify a request type. Availability of Request Types is dependent upon role configuration. For Standard and Temporal Requests, the request types are:		Required
			SDR_EDR_IP	These include SDRs, EDRs, TDRs, ARPs, and IPs. Which one of these is desired is controlled by the category parameter.	
			DIARY	These include Diary, Dump, and Dwell RDR that give various information about spacecraft health, attitude, and ephemeris.	
			RDR	These include science, diagnostic RDRs.	
			GRIDDED_IP	Gridded Intermediate Products.	
			CATALOG	Catalog Requests for products on the system catalog.	
			AUX	Auxiliary Data. These data products are produced by NPOESS, and are required by NPOESS algorithms (with the exception of sensor data) to achieve specific system specification performance attributes.	
			ANC	Ancillary Data. These data products are not produced by NPOESS, but are required by NPOESS algorithms to meet the attributes specified by the system specification for applicable NPOESS Data Products.	
Request Name	All	All	The user supplies a request name. This name should be unique to distinguish it from other requests. If the user submits multiple requests with the same Request Name, they will be given separate Request IDs and be treated as separate requests.		Required

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Request ID	All	All	The system generates a unique Request when an ID is requested or when a request without a Request ID is submitted. For all other actions on the request, the user supplies this parameter.	Required, except for request creation
Destination	Standard, Temporal, Periodic, and Catalog Request	All	The user specifies the destination of the requested data. Delivery of the data may require a username and password for some destinations. Prior to submitting the data request, the user must supply a destination. Prior to shipping the data, the software checks to see that the destination is on the destination list maintained by the software. The destination may be modified, added to, or deleted. There may be more than one destination associated with a request, and there must always be at least one destination associated with a request.	Required

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Effectivity	Standard, Temporal, Periodic, and Catalog Request	All	Effectivity specifies the window of interest that the request is effective for. For Standard and Periodic Requests, this refers to the observation time (NPP/NPOESS Data Products) or effectivity time, as applicable, of the data to be delivered. Periodic Requests will have a start time but no end time. Periodic Requests are assumed to run for the lifetime of the program. The user must specify an effectivity start time, but can choose not to specify an end time. Time is referenced to IET. For information on IET see JPSS CDFCB-X Vol. I, 474-00001-01. The minimum start time is 949388400000000 (1988-02-01). The maximum start time (or end time) is 5680281632999999 (2137-12-12 23:59:59). It is possible for the user to specify effectivity in such a way that no data will be delivered. for example, if both start and stop times are more than 24 hours in the past the user can submit: • A standing request. Standing requests have no end time specified. Any data that meets the criteria after the start time will be sent. • A one time request. For these requests, a start time and end time are required. End times can be set to a point in the future up to 1,167,696,000 seconds.	Required
Processing Delay	Standard and Temporal	SDR/EDR/IP, RDR, DIARY	If desired, the user may specify a processing delay in microseconds. A processing delay of 3,600,000,000 microseconds causes the system to wait 1 hour after data observation time, before the data is delivered. Only the latest version of a granule is delivered. Maximum Processing Delay is 43,200,000,000 microseconds.	Optional

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Package Data According to Packaging Rules	Standard and Temporal	SDR/EDR/IP	If desired, the user can request delivered data to be packaged with its corresponding GEO products or it can be delivered separately. For example if a user is requesting VIIRS-I1-SDR, if packaging is selected the user would receive the VIIRS-IMG-GEO secondary product in one HDF file: SVI01-GIMGO_npp…	Optional
Deliver Repaired Data	Standard and Temporal	SDR/EDR/IP, RDR, DIARY	If desired, the user can request repaired data be delivered if it becomes available after the request is initially filled, but before the request expires. The repair delivery will be in accordance with the packaging option of the request, but unaggregated (even if the initial delivery was aggregated). The repair delivery will include the repaired granule and its associated primary/secondary granules.	Optional
Temporal	Temporal	SDR/EDR/IP, RDR, ANC, AUX, DIARY	The temporal subsetting is specified by a start day, end day, start time, and duration. Start/End day is referenced using IET in UTC. Start time and duration are provided in microseconds. Both values must be less than 86,400,000,000 microseconds (one day). Temporal subsetting provides the time period during each day that data will be provided. If a request has start/end dates that span ten days and a start time of 0700 for a duration of 6 hours, then data collected between 0700 and 1300 UTC for each of the ten days will be sent. Data is only delivered if it is within the temporal subset time. The delivered data might be slightly larger than the requested data because the delivery includes the entire granule that contains the specified time.	Optional

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Geospatial Subsetting	Standard, Temporal, and Catalog Query	SDR/EDR/IP, RDR,	The user may request NPP/NPOESS Data Products by specifying the upper left (north and west of the center of the region) and lower right (south and east of the center of the region) geographic coordinates. This bounding rectangle is a footprint on the ground only and is not related to the direction of the orbit or to the data collection parameters. The bounding rectangle can cover an area that contains data collected in more than one orbit. The coordinates may be specified using degrees, minutes, and seconds, or by using decimal degrees. Latitude ranges from positive (north of the Equator) 90 degrees to negative (south of the Equator) 90 degrees. Longitude ranges from positive (east of the Prime Meridian) 180 degrees to negative (west of the Prime Meridian) 180 degrees. The delivered data might be slightly larger than the requested data because the delivery includes the entire granule that contains the specified coordinates. There exists internal limits on how small or large the geospatial region can be. The maximum distance between the two pairs of coordinates is 15,000 km. The distance is not checked at request submission time. A request with geospatial subsetting that exceeds these limits will be failed by DDS and an appropriate system message will be generated to describe the nature of the error.	Optional

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Aggregation	Standard and Temporal	SDR/EDR/IP, RDR, DIARY	A user may receive data in individual granules or they may have granules aggregated into one file. An aggregation is a collection of granules of a particular NPP/NPOESS Data Product over time. If the user specifies aggregated granules, the user must provide an aggregation period greater than 1 second (1000000) and less than maximum aggregation size, nominally 104 minutes (6240000000). The number of granules per HDF5 file is calculated as the ceiling of the aggregation period divided by the granule size. The resulting HDF5 file(s) will contain the total number of granules based on the aggregation size. The alignment of the HDF5 aggregations is based on the first ascending node after launch, which may yield fewer granules in the aggregations at either or both request boundaries. If no aggregation period is provided, the user will receive each granule that meets the request criteria in a separate file. Aggregation is specified on a per request basis. For a detailed description of the calculations involved, refer to the JPSS CDFCB-X Vol. I, 474-00001-01.	Optional
Data Product ID	Standard, Temporal, Periodic, Catalog Request and Catalog Query	SDR/EDR/IP, RDR, DIARY, ANC, and AUX	The ID of the data product to be delivered. A request may contain many data product IDs. This definition is specific to the API requests only and is not the same as Data Product Id specified in other program documents. The DataProductID parameter is an identifier that is synthesized by merging a Collection Short Name and Spacecraft Name. For a complete list of Collection Short Names, see the JPSS CDFCB-X Vol. I, 474-00001-01. The Spacecraft Name is either the Mission Name (for ANC products) or the Platform Short Name (for all other products). A complete list of Platform Short Names and Mission Names can be found in the JPSS CDFCB-X Vol. V, 474-00001-05. Each data product also has a sensor associated with it, but it is not formally part of the Data Product ID.	Required

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Orbit ID	Standard and Temporal	SDR/EDR/IP, RDR, DIARY	The range of revolution (orbit) numbers of the data to be delivered. The range is specified by a start orbit and an end orbit, inclusively. Only the data that matches both the orbit numbers and the effectivity parameters will be delivered. Please note that for the Orbit ID parameter to be used effectively it requires external knowledge of the times at which a given orbit will/has occurred.	Optional
Periodicity	Periodic	GRIDDED_IP	The user can specify the period at which the request will run and deliver new/updated files. This value is specified as a number of days, hours, minutes, and seconds. If this parameter is not set, the Periodic request will run only once, delivering all available files, and then it will expire.	Optional

4.3.2 Request Examples

The following sections provide examples for specifying common and unique user-defined parameters for a variety of request types, including Standard and Temporal Requests for NPP/NPOESS Data Products, Ancillary Data, and Auxiliary Data.

4.3.2.1 Specifying a Standard Request for SDR/EDR/IP/TDRs

A Standard Request is a request for a data record, such as an ARP or an EDR. The tables in this section represent examples of requests for NPP/NPOESS Data Product types. Table 4.3.2.1-1, Standard Request Parameters for SDR/EDR/IP/TDRs, identifies parameters required to submit a request for an EDR from the VIIRS sensor on the NPP satellite. The effectivity is for one week; beginning 25 Jan 200316 May 2008 07:00:0023:05:00 – 25 Jan 2003 08:00:00. Aggregation is 10 minutes. Processing Delay is set to 5 minutes. The request will deliver repaired data and package data according to packaging rules for data that is located in those GEO coordinates that are supplied with the request.

Table 4.3.2.1-1, Standard Request Parameters for SDR/EDR/IP/TDRs

Standard Request	SDR/EDR/IP/TDR	Request Name	MyRequest
		Request ID (Server Generated)	100000000001
		Data Product ID	VIIRS-CM-IP_NPP
		Destination	100000000045
		Effectivity Start/End Time	1422169232000000
			1422172832000000
		Aggregation	600000000
		Processing Delay	300000000
		Geospatial Subsetting On	Upper Left Latitude 46.10
			Upper Left Longitude -145.30
			Lower Right Latitude -45.10
			Lower Right Longitude -145.10
		Deliver Repaired Data	y
		Package data according to packaging rules	y

4.3.2.2 Specifying a Standard Request for RDRs

A Standard request is a request for a raw data record. The tables in this section represent examples of requests for RDR type products (not Diary RDR). Table 4.3.2.2-1 Standard Request Parameters for RDRs, identifies parameters required to submit a request for an RDR from the ATMS sensor on the NPP satellite. The effectivity is for one week; beginning 25 Jan 2003 07:00:00 – 25 Jan 2003 08:00:00 UTC. Aggregation is 10 minutes. Processing Delay is set to 5 minutes. The request will deliver repaired data for data that is located in those GEO coordinates that are supplied with the request.

Table 4.3.2.2-1 Standard Request Parameters for RDRs

Standard Request	RDR	Request Name	MyRequest
		Request ID (Server Generated)	100000000001
		Data Product ID	ATMS-SCIENCE-RDR_NPP
		Destination	100000000045
		Effectivity Start/End Time	1422169232000000
			1422172832000000
		Aggregation	600000000
		Processing Delay	300000000
		Geospatial Subsetting On	Upper Left Latitude 46.10
			Upper Left Longitude -145.30
			Lower Right Latitude -45.10
			Lower Right Longitude -145.10
		Deliver Repaired Data	y

4.3.2.3 Specifying a Standard Request for Spacecraft Diary, Telemetry, Dwell, and Dump RDRs

A Standard request is a request for a raw data record. The tables in this section represent examples of requests for Diary/Dump/Dwell type products. Table 4.3.2.3-1 Standard Request Parameters for Spacecraft Diary RDRs, identifies parameters required to submit a request for an RDR from the ATMS sensor on the N01 satellite. The effectivity is for one hour; beginning 25 Jan 2003 07:00:00 – 25 Jan 2003 08:00:00 UTC. Aggregation is 10 minutes. Processing Delay is set to 5 minutes. The request will deliver repaired data.

Table 4.3.2.3-1 Standard Request Parameters for Spacecraft Diary RDRs

Standard Request	Diary	Request Name	MyRequest
		Request ID (Server Generated)	100000000001
		Data Product ID	ATMS-DUMP-RDR_N01
		Destination	100000000045
		Effectivity Start/End Time	1422169232000000
			1422172832000000
		Aggregation	600000000
		Processing Delay	300000000
		Deliver Repaired Data	y

4.3.2.4 Specifying a Temporal Request for SDR/EDR/IP/TDRs

Table 4.3.2.4-1, Temporal Request Parameters shows the parameters used to temporally request an IP for the VIIRS Cloud Mask product, with a start day of 20 Jan 2003 – 30 Jan 2003 with a start time of 20 Jan 2003 at 07:00:00 with a 6 minute duration. This is for data coming from the NPP platform. The geospatial setting is set.

Table 4.3.2.4-1, Temporal Request Parameters

Temporal Request	SDR/EDR/IP/TDR	Request Name	MyRequest
		Request ID (Server Generated)	100000000001
		Data Product ID	VIIRS-CM-IP_NPP
		Destination	100000000045
		Effectivity	**Start Day:** 1421712032000000
			End Day: 1422576032000000
			Start Time: 1421737232000000
			Duration: 21600000000
		Aggregation	600000000
		Processing Delay	300000000
		Geospatial Subsetting On	Upper Left Latitude 46.10
			Upper Left Longitude -145.30
			Lower Right Latitude -45.10
			Lower Right Longitude -145.10
		Deliver Repaired Data	y

		Package data according to packaging rules	y

4.3.2.5 Specifying Standard Requests for Ancillary Data

Ancillary Data requests are requests for data not produced by NPOESS, but required by NPOESS algorithms to meet the attributes given in the System Specification (SY15-0007) Appendix D (e.g., terrain height database or conventional surface and upper air observations).

The Ancillary Data request shown in Table 4.3.2.5-1, Ancillary Data Standard Request Parameters shows the parameters used to request Ancillary Data of type NCEP-GFS-12HR-ANC to be delivered to the user. The effectivity is for one hour; beginning 25 Jan 2003 07:00:00 – 25 Jan 2003 08:00:00 UTC.

Table 4.3.2.5-1, Ancillary Data Standard Request Parameters

Standard Request	**Ancillary**	**Request Name**	MyRequest
		Request ID (Server Generated)	100000000001
		Data Product ID	NCEP-GFS-12HR-ANC_NPP
		Destination	100000000045
		Effectivity Start/End Time	1422169232000000
			1422172832000000

4.3.2.5.1 Ancillary Data Request Behavior

For a standard ancillary data request the data product ID and effectivity parameters will determine which files to deliver. All ancillary files that match the data product ID and overlap the effectivity (partially or completely) will be delivered. The ancillary data request does not allow for a "deliver repair data" parameter, this is due to the nature of the ancillary data itself. Ancillary data is not formally repaired, although updates for ancillary data may be generated within IDPS that overlap (in effectivity) previously generated ancillary data. Due to this, the ancillary data request will deliver all ancillary

files that match the parameters, some of which may overlap each other in effectivity.

4.3.2.6 Specifying Standard Requests for Auxiliary Data

Auxiliary Data requests are requests for data produced by NPOESS, other than sensor data, which are required by NPOESS algorithms to achieve the performance attributes given in the System Specification (SY15-0007) Appendix D (e.g., ephemeris data, sensor calibration coefficients, sun angles). Auxiliary Data is identified by their Collection Short Names.

The Auxiliary Data request shown in Table 4.3.2.6-1, Auxiliary Data Standard Request Parameters shows the parameters used to request Auxiliary Data of type TLE_AUX to be delivered to the user. The effectivity is for one hour; beginning 25 Jan 2003 07:00:00 – 25 Jan 2003 08:00:00 UTC.

Table 4.3.2.6-1, Auxiliary Data Standard Request Parameters

Standard Request	**Auxiliary**	**Request Name**	MyRequest
		Request ID (Server Generated)	100000000001
		Data Product ID	TLE-AUX_NPOESS
		Destination	100000000045
		Effectivity Start/End Time	1422169232000000
			1422172832000000

4.3.2.6.1 Auxiliary Data Request Behavior

For a standard auxiliary data request, the data product ID and effectivity parameters will determine which files to deliver. All auxiliary files that match the data product ID and overlap the effectivity (partially or completely) will be delivered. The auxiliary data request does not allow for a "deliver repair data" parameter, this is due to the nature of the auxiliary data itself. Auxiliary data is not formally repaired, although updates for auxiliary data may be generated within IDPS that overlap (in effectivity) previously generated auxiliary data. Due to this, the auxiliary data request will deliver all auxiliary files that match the parameters, some of which may overlap each other in effectivity.

4.4 Request Templates

Templates are exact copies of previously submitted requests that can be used to re-submit or create requests without having to enter every field. New requests can be created from templates, then modified and submitted. New templates can be created from existing requests as well. This class of user functions also includes viewing and deleting templates.

4.5 Destination List Management

The user must have a valid destination before they can successfully submit a request through the API. Destination List Management functions allow the user to view the list of destinations, add new destinations, modify existing destinations, or delete existing destinations.

4.6 Catalog Management

This class of functions has two sub-categories:

1) Those functions that allow the user to query the catalog for a list of available data and filter and view the resulting list. These are referred to as Data Catalog Queries.

2) Those functions that allow the user to request a particular piece of data from the catalog. These are referred to Data Catalog Requests. The user must query the catalog in order to obtain the information necessary to request data.

4.6.1 Data Catalog Queries

The catalog query functions allow the user to create queries of the catalog's content and to view the results of those queries. It also has functions that allow the user filter the results, to view the existing queries, or to delete a query. The request functions are a subset of the general request functions described above.

A catalog query is a request for a list of granules that are in the system and available to be shipped to the user. The user can specify that the list be filtered by request type, product type or category, collection short name, spacecraft, or sensor. The URIDs returned as part of the query results are needed as inputs to Data Catalog Requests.

4.6.2 Data Catalog Requests

The second class of functions allows the user to create catalog requests to retrieve the actual data from storage and have it delivered to them. The user can view single requests, view all of the existing requests, create new requests, or delete the requests.

Catalog requests are made for a single, existing item of data only and no aggregation is available. Packaging is always turned on (i.e. data products are always delivered with the geolocation). These products are identified with unique ID numbers. The URID must be known prior to making this request and may be found by examining the results of a catalog query. Table 4.6.2-1, Data Catalog Request Parameters, shows the parameters used for a Data Catalog Request, while Table 4.6.2-2, Data Catalog Request Parameters Example, identifies common and user-defined parameters required to submit requests for Data Catalog products.

Table 4.6.2-1, Data Catalog Request Parameters

Common User-Supplied Parameters	Applicable Request Type	Description	Required / Optional
Request Name	All	The user supplies a request name. This name should be unique to distinguish it from other requests. If the user submits multiple requests with the same Request Name, they will be given separate Request IDs and be treated as separate requests.	Required
Request ID	All	The system generates and returns a unique Request ID to the user when an ID is requested or when a request without a Request ID is submitted. For all other actions on the request, the user supplies this parameter.	Required, except for request creation
URID	Data Catalog	URID also known as UR ID or Universal Reference ID is a unique identifier for a data product. This is sometimes also referred to as the GranuleID in the code signatures. URID is defined as N_Reference_ID in the JPSS CDFCB-X Vol. V, 474-00001-05.	Required

Table 4.6.2-2, Data Catalog Request Parameters Example

Data Catalog Request	**Catalog**	**Request Name**	MyCatalogRequest
		Request ID (Server Generated)	100000000001
		URID	43132603-11104-9b9dea63-deb2216a
		Destination	100000000045
		Effectivity Start/End Time	1422169232000000
			1422172832000000

Request Name

Request ID (Returned) 98765

URID 43132603-11104-9b9dea63-deb2216a

Destination h:/fileCatalog

4.7 Supervisor Functions

Some users are given supervisory duties for a set of sub-users. These supervisors have functions available to them that they can use to view, suspend, resume, or delete requests for other users. They may also transfer ownership of requests from one user to another, as long as the user's roles are the same for both the original request owner and the new request owner.

5. DELETED

6. C++ API DOCUMENTATION

The C++ API is a set of libraries that were developed and tested on a Microsoft® Windows® or IBM AIX® platform. The user application is defined to be the application that is using the API. The central object of the API is the Message object. On IBM AIX®, all NPOESS API classes are in the libDDSAPI.a library if the static library is used and libDDSAPI.so library if the shared library is used. On Microsoft® Windows®, only the static library, libDDSAPI.lib, is available. There should only be one Message object instantiated for the user application. The default constructor should be used to construct the Message class in most cases. The default constructor assumes that the environment variables DDS_ PORT, DDS_ HOST, DDS_TIMEOUT, DSTATICDATA, INFUTIL_CFGDOMAIN, HTTPS_HOST, and HTTPS_PROTOCOL are already defined.

6.1 Coding Conventions

The coding conventions used for the NPOESS API comply with the NPOESS Software Standard and Practices Manual, MN60822-PMO-001.

6.1.1 C++ Coding Conventions

All C++ files follow the conventions found in Table 6.1-1, C++ Coding Conventions.

Table 6.1-1, C++ Coding Conventions

C++ Coding Conventions	
Application	**Convention**
Constants	All constants meant for users of the class are static and are publicly accessible.
Non-Static Variables	Non-static variables are private and may only be accessed through setters and getters. If a setter or getter doesn't exist, then that variable cannot be accessed in that fashion.
Classes	All classes have a default constructor. Some of those default constructors may not be accessible to the user application, since those classes should not be instantiated in that manner.

C++ Coding Conventions	
Application	**Convention**
Error exception reporting	All errors will be reported as a C++ exception. There are three types of exceptions that may be thrown, including: DDSAPI_RequestException DDSAPI_MessageException DDSAPI_Exception DDSAPI_Exception is the base exception class for the other two, so simply catching DDSAPI_Exception when calling methods that throw any of these three exceptions works.
==() method	==() method exists in classes that could be compared; in particular, this applies to data product and request related data.
Copying Objects	Certain objects can be copied. Those objects implement a copy constructor and an operator. Those objects perform a deep copy to ensure all data is copied. These are for data product and request-related data.
Memory Management	All requests, templates, data products, and catalogs are managed through the Message object. **Do not attempt to create your own and add it. These objects will be created and destroyed by the API.** The Message object needs to be instantiated and deleted by the user application.

Refer to the NPOESS Software Standard and Practices Manual (SSPM), MN60822-PMO-001, Appendix E, for additional details on coding guidelines.

6.1.2 Environment Variables

Following is a list of the environment variables that need to be specified for use by the C++ Client.

The **DDS_PORT** environment variable tells the C++ API application what port to use when communicating with the DDS Server.
The **DDS_HOST** environment variable tells the C++ API application the IP address of the machine that is hosting the DDS Server.
The **DDS_TIMEOUT** variable tells the C++ API how many minutes of inactivity before closing the session.
The **DDS_ROOT** variable identifies the location where IDPS log files should be stored if necessary.
The **DINFUTIL_CFGDOMAIN** environment variable tells the C++ API application the location of the "INF_GuideList.cfg" file.
The **DHTTPS_PROTOCOL** variable tells the C++ API application if HTTPS_PROTOCOL is either http or https
The **DHTTPS_HOST** variable tells the C++ API application the IP address of the host that is running the Apache Web Server
The **DDS_HEARTBEAT_DURATION** variable tells the C++ API application the duration in seconds when the Server will time out and disconnect the API if the heartbeat is not received.
The **VBROKERDIR** variable identifies the location of Visibroker COTS lib files

Notes:
The IDPS Windows Installshield creates this environment variable when it's installed

6.1.3 Procedures for Client-side DDS API SSL Certificate Installation

TBD – C++ API Certificate Installation procedures are still in work.

6.2 C++ API Module Documentation List

The C++ API consists of a set of classes defining the attributes, enumerations, and functions that allow the user to logon, create a request, process a request, and perform the normal manipulations on catalog items and templates.

6.2.1 DDSAPI_Message Class Reference

This object is responsible for establishing and maintaining contact with the API Manager. It is also responsible for handling commands that can be performed in the system. The various API commands are executed as calls on methods in this class. Most pointers return a copy of memory referenced by the API. It should be deleted by the caller. The caller keeps ownership to the pointers passed in. The API does not delete the data passed in. A user of the API must create an instance of this class to interact with the API. All interaction with the API should be done through this object or objects returned by this object. After this class is created a user of this class must login to the API to use it. Calls may then be made on public methods in this class or in the classes returned by these

methods. If there are problems executing the methods in the API then a message will be created and added to the System Messages. A user can then use the getSystemMessages() call to get the current system messages. A method will exit with a false, 0, or an empty vector if there is a problem with the API usage. Under normal API operation an exception should not be passed back to the caller of any API method. Do not try to create objects outside of this class. The API will not know about them or be able to use them.

The Class diagram representing the DDSAPI_Message Class is provided in Figure 6.2.1-1, DDSAPI_Message Class UML Diagram.

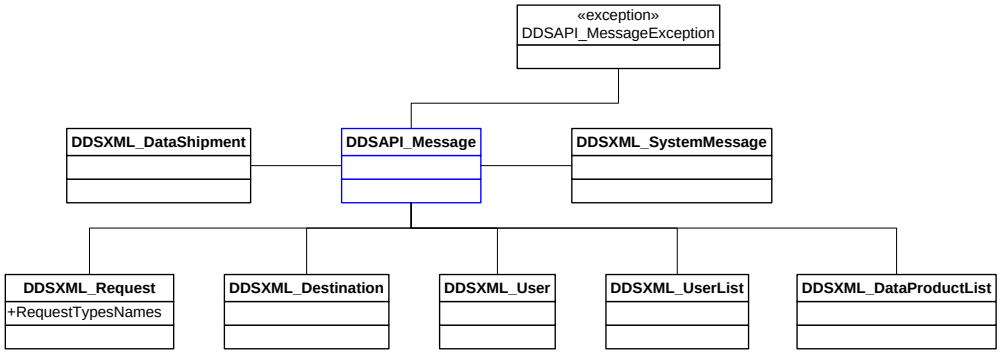


Figure 6.2.1-1, DDSAPI_Message Class UML Diagram

6.2.1.1 DDSAPI_Message Class Attributes

static std::string DEFAULT_HOST – The default environmental variable that stores the host name to connect to.

static std::string DEFAULT_PORT – The default environmental variable that stores the port number to connect to.

static int DEFAULT_TIMEOUT – The default time out for the Message.

6.2.1.2 DDSAPI_Message Class Functions

6.2.1.2.1 DDSAPI_Message::DDSAPI_Message

DDSAPI_Message::DDSAPI_Message (

std::string hostname =

DEFAULT_HOST,

std::string port = DEFAULT_PORT,

int timeout = DEFAULT_TIMEOUT, DDSXML_Command::DDSXML_ClientTypes

clientType = DDSXML_Command::CLIENT_API,

const std::string & httpsURL = "",

bool useHTTPSFlag = false)

throw (

DDSAPI_MessageException

)

Overloaded Constructor. This constructor must be used to create the DDS API interface to the DDS Server. The default constructor should not be used. The proper host and port must be used that matches the DDS Server setup. Make sure that these are configurable as they may change for different Server connections. This should be the only case in this object where an exception will be thrown since we can not create an instance of this object.

Parameters:

hostname The environmental variable to retrieve the host name of the Request Server from.

port The environmental variable to retrieve the port number from.

timeout The timeout duration in seconds.

clientType The client type to create - CLIENT_API

httpsURL The https url string, Example: "https://auisdev26/secure-server/services/SecureServerAPI"

useHTTPSFlag - True to use https

Exceptions:

DDSAPI_MessageException if initialization of infrastructure fails

6.2.1.2.2 DDSAPI_Message::~DDSAPI_Message

DDSAPI_Message::~DDSAPI_Message (

)

Destructor

6.2.1.2.3 DDSAPI_Message::login

bool DDSAPI_Message::login (

std::string username,

std::string password,

std::string role

)

This method provides a mechanism for user login into the system. The user must provide a username, and password. The role parameter is optional if this user only has one role. If authentication fails, false will be returned to the caller. The API must also be configured to use the API calls.

Parameters:

username The username to be used to authenticate the user.

password The password associated with the username.

role The role that is associated with the username for this login.

Returns:

bool true - if the user has successfully logged into the system. false - if the user login attempt has failed.

6.2.1.2.4 DDSAPI_Message::logout

bool DDSAPI_Message::logout (

)

Sends a message to the server that the API is ready to log out the user. The session is only disconnected on timeout or if this object is deleted. This must be called when you are done using the API. The Server uses this to clean up internal memory faster.

Returns:

bool true - if the logout was successful false - if the logout attempt failed

6.2.1.2.5 DDSAPI_Message::getLoginState

bool DDSAPI_Message::getLoginState (

)

This method returns the login state of the system. Make sure to look at the configuration state also.

Returns:

bool The login state, true - logged in false - no logged in

6.2.1.2.6 DDSAPI_Message::getConfigState

bool DDSAPI_Message::getConfigState (

)

This method returns the config state of the system. If true then the API has been fully configured. If false, then a part or all of the API has not been configured. The API must be fully configured to use the other methods in the API.

Returns:

bool The config state, true - configured false - not configured

6.2.1.2.7 DDSAPI_Message::getRoles

std::vector<std::string> DDSAPI_Message::getRoles (

std::string username,

std::string password

)

This method returns a vector of strings that contain all of the user's valid roles as strings. This is the only API method that does not require the user to be logged in before it can be called. This is to allow a display/GUI to show a user all of their roles when logging into the DDS Server.

Parameters:

username The user's username whose roles are to be obtained from the Request Server.

password The user's password whose roles are to be obtained from the Request Server.

Returns:

std::vector<std::string> The vector of strings that contain the user's possible roles.

6.2.1.2.8 DDSAPI_Message::getSystemMessages

std::vector<DDSXML_SystemMessage*> DDSAPI_Message::getSystemMessages (

DDSXML_User * user = 0

)

This method returns a vector of DDSXML_SystemMessage pointers to the user. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

user The user that is requesting this API call.

Returns:

std::vector<DDSXML_SystemMessage*> The vector of DDSAPI_SystemMessages received by the DDSAPI_Message.

6.2.1.2.9 DDSAPI_Message::getStoredSystemMessages

std::vector<DDSXML_SystemMessage*>

DDSAPI_Message::getStoredSystemMessages (

DDSXML_User * user = 0

)

This method returns a vector of DDSXML_SystemMessage pointers to the user. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

user The user that is requesting this API call.

Returns:

std::vector<DDSXML_SystemMessage*> The vector of DDSAPI_SystemMessages received by the DDSAPI_Message.

6.2.1.2.10 DDSAPI_Message::getDataShipments

std::vector<DDSXML_DataShipment*> DDSAPI_Message::getDataShipments (

std::string requestID,

DDSXML_User * user = 0

)

This method returns a vector of DDSXML_DataShipment pointers to the user. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

requestID The ID of the DDSXML_Request to get the shipment records for.
user The user that is requesting this API call.

Returns:

std::vector<DDSXML_DataShipment*> The vector of DDSXML_DataShipment received by the API.

6.2.1.2.11 DDSAPI_Message::addDestination

```
bool DDSAPI_Message::addDestination (
    std::string destinationName,
    std::string hostName,
    std::string path,
    std::string username,
    std::string password,
    DDSXML_Destination::DDSXML_DestinationTransferType transferType,
    DDSXML_User * user = 0
  )
```

This method adds a new destination to the user defined destination list.

Parameters:

destinationName The name for this destination.
hostName The hostname or IP address.
path The destination path.
username The username for the destination FTP server.
password The password for the destination FTP server.
transferType The transferType for the destination.

user The user that is requesting this API call.

Returns:

bool True if added OK to the Server, false if not.

6.2.1.2.12 DDSAPI_Message::addNewDestination

std::string DDSAPI_Message::addNewDestination (

std::string destinationName,

std::string hostName,

std::string path,

std::string username,

std::string password,

DDSXML_Destination::DDSXML_DestinationTransferType transferType,

DDSXML_User * user = 0

)

This method adds a new destination to the user defined destination list.

Parameters:

destinationName The name for this destination.

hostName The hostname or IP address.

path The destination path.

username The username for the destination FTP server. Not required for a local destination

password The password for the destination FTP server. Not required for a local destination

transferType The transferType for the destination.

user The user that is requesting this API call.

Returns:

std::string Destination ID or "" if there was an error.

6.2.1.2.13 DDSAPI_Message::deleteDestination

bool DDSAPI_Message::deleteDestination (

DDSXML_Destination * destination,

DDSXML_User * user = 0

)

This method removes a user defined destination from the user destination list. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

destination The destination to be removed from the destination list.

user The user that is requesting this API call.

Returns:

bool True - if successful, False - if failed

6.2.1.2.14 DDSAPI_Message::getDestinations

std::vector< DDSXML_Destination*> DDSAPI_Message::getDestinations (

DDSXML_User * user = 0

)

This method returns all of the user's defined destinations in a vector to the caller. These destinations are owned by the caller and must be deleted. The caller still owns the pointer passed in. The API will not delete the pointer. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

user The user that is requesting this API call.

Returns:

std::vector< DDSXML_Destination*> The vector containing all user-defined destinations.

6.2.1.2.15 DDSAPI_Message::getDestination

virtual DDSXML_Destination* DDSAPI_Message::getDestination (

std::string userIndex,

DDSXML_User * user = 0

)

This method returns the destination identified by the destination ID if it exists in the system The caller still owns the pointer passed in. The API will not delete the pointer. The pointer point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

userIndex The userIndex to check for

user The user that is requesting this API call.

Returns:

DDSXML_Destination* A pointer to the destination . 0 - The destination was not found. Valid pointer otherwise.

6.2.1.2.16 DDSAPI_Message::modifyDestination

bool DDSAPI_Message::modifyDestination (

DDSXML_Destination * destination,

DDSXML_User * user = 0

)

This method modifies the user destination. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

destination The user defined destination to be modified.

user The user of this command

Returns:

bool True - if successful, False - if failed

6.2.1.2.17 DDSAPI_Message::getDataProductList

DDSXML_DataProductList* DDSAPI_Message::getDataProductList (

DDSXML_User * user = 0

)

This method returns a DDSXML_DataProductList object to the user that contains all the

possible data products that the user application may request based on the username and role. The user may then filter data products using the DataProductList object. The caller still owns the pointer passed in. The API will not delete the pointer. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

user The user that is requesting this API call.

Returns:

DDSXML_DataProductList* The pointer to the data product filter list

6.2.1.2.18 DDSAPI_Message::getFilteredDataProductList

```
DDSXML_DataProductList* DDSAPI_Message::getFilteredDataProductList (
    DDSXML_User * user,
    DDSXML_Request::DDSXML_RequestTypes requestType
  )
```

This method returns a DDSXML_DataProductList object to the user that contains all the possible data products that the user application may request based on the username and role, restricted by those applicable to the supplied request type

Parameters:

user The user that is requesting this API call.

requestType The request type against which the products will initially be filtered

Returns:

DDSXML_DataProductList The pointer to the data product filter list

6.2.1.2.19 DDSAPI_Message::createRequest

```
DDSXML_Request* DDSAPI_Message::createRequest (
    DDSXML_Request::DDSXML_RequestTypes requestType =
    DDSXML_Request::SDR_EDR_IP_REQUEST_TYPE,
    DDSXML_Request::DDSXML_ImplementationRequestTypes implType =
    DDSXML_Request::DEFAULT_IMPL_REQUEST_TYPE,
```

DDSXML_User * user = 0

)

This method instantiates a Request and returns a pointer to the instantiation. This is owned by the caller and should be deleted by the user. This must be added to the Server using addRequest. The caller still owns the pointer passed in. The API will not delete the pointer. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

requestType The request type to use in this API call
implType The implType type to use in this API call
user The user that is requesting this API call.

Returns:

DDSXML_Request* A pointer to the newly constructed DDSXML_Request

6.2.1.2.20 DDSAPI_Message::createTemplateFromRequest

DDSXML_Request* DDSAPI_Message::createTemplateFromRequest (

std::string requestID,

DDSXML_User * user = 0

)

This method creates a Template using data from an existing request and returns a pointer to the instantiation. This is owned by the caller and should be deleted by the user. This must be added to the Server using addRequest. The caller still owns the pointer passed in. The API will not delete the pointer. The pointer point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

requestID The ID of the template to be copied.
user The user that is requesting this API call.

Returns:

DDSXML_Request* A pointer to the newly constructed DDSXML_Request

6.2.1.2.21 DDSAPI_Message::createRequestFromTemplate

DDSXML_Request* DDSAPI_Message::createRequestFromTemplate (

std::string requestID,

DDSXML_User * user = 0

)

This method creates a Request using data from an existing template and returns a pointer to the instantiation. This is owned by the caller and should be deleted by the user. This must be added to the Server using addRequest. The caller still owns the pointer passed in. The API will not delete the pointer. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

requestID The ID of the template to be copied.
user The user that is requesting this API call.

Returns:

DDSXML_Request* A pointer to the newly constructed DDSXML_Request

6.2.1.2.22 DDSAPI_Message::addRequest

bool DDSAPI_Message::addRequest (

DDSXML_Request * request,

DDSXML_User * user = 0

)

This method adds the request to the Request Server. This can be any request, Template or modified request. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

request The request to be added to the Server.
user The user that is requesting this API call.

Returns:

bool - True if submitted OK, False if not

6.2.1.2.23 DDSAPI_Message::findRequest

```
DDSXML_Request* DDSAPI_Message::findRequest (
    std::string requestID, DDSXML_Request::DDSXML_ImplementationRequestTypes
    implType,
    DDSXML_Request::DDSXML_RequestTypes requestType =
    DDSXML_Request::UNKNOWN_REQUEST_TYPE,
    bool templateFlag = false,
    DDSXML_User * user = 0
  )
```

This method returns the Request* that references the request that had a request ID equal to requestID. The caller still owns the pointer passed in. The API will not delete the pointer. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

requestID The request ID for the request to be found.
implType The implType type to use in this API call
requestType The request type to use in this API call
templateFlag True to verify if this is a template
user The user that is requesting this API call.

Returns:

DDSXML_Request* a pointer to the request found in the system that matches the ID passed in.

6.2.1.2.24 DDSAPI_Message::getRequests

```
std::vector<DDSXML_Request*> DDSAPI_Message::getRequests (
    DDSXML_Request::DDSXML_ImplementationRequestTypes implType,
    DDSXML_Request::DDSXML_RequestTypes requestType =
    DDSXML_Request::UNKNOWN_REQUEST_TYPE,
```

bool templateFlag = false,

bool allRequests = false,

DDSXML_User * user = 0

)

This method will return all Requests for the user as a vector of Request*. The caller still owns the pointer passed in. The API will not delete the pointer. The pointers point to memory that is owned by the caller and it is the responsibility of the user to free this memory.

Parameters:

implType The implType type to use in this API call
requestType The request type to use in this API call
templateFlag True to verify if this is a template
allRequests True to add in all request
user The user that is requesting this API call.

Returns:

std::vector<DDSXML_Request> The vector of strings that contain the user's possible roles.

6.2.1.2.25 DDSAPI_Message::suspendRequest

bool DDSAPI_Message::suspendRequest (

std::string requestID,

Int64 duration = -1,

DDSXML_User * user = 0

)

This method suspends the request referenced by the requestID. The user may also provide a duration. If no values are given then all data products within the request are suspended indefinitely. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

requestID The ID of the request to be suspended.

duration The duration for the suspension in seconds. If this value is -1, the data product(s) will be suspended indefinitely.

user The user that is requesting this API call.

Returns:

bool True - if successful, False - if failed

6.2.1.2.26 DDSAPI_Message::resumeRequest

```
bool DDSAPI_Message::resumeRequest (
    std::string requestID,
    DDSXML_User * user = 0
  )
```

This method resumes data products specified by the requestID. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

requestID The ID of the request to be resumed.

user The user that is requesting this API call.

Returns:

bool True - if successful, False - if failed

6.2.1.2.27 DDSAPI_Message::deleteRequest

```
bool DDSAPI_Message::deleteRequest (
    std::string requestID,
    DDSXML_User * user = 0
  )
```

This method deletes a request from the system based on the request ID given as a parameter. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

requestID The ID of the request to be deleted.

user The user that is requesting this API call.

Returns:

bool True - if successful, False - if failed

6.2.1.2.28 DDSAPI_Message::deleteAllRequests

bool DDSAPI_Message::deleteAllRequests (

DDSXML_User * user

)

This method deletes all requests from the system for the logged in user. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

user The user that is requesting this API call.

Returns:

bool True - if successful, False - if failed

6.2.1.2.29 DDSAPI_Message::getNumberOfRequests

int DDSAPI_Message::getNumberOfRequests (

DDSXML_Request::DDSXML_ImplementationRequestTypes implType,

DDSXML_Request::DDSXML_RequestTypes requestType =
DDSXML_Request::UNKNOWN_REQUEST_TYPE,

bool templateFlag = false,

DDSXML_User * user = 0

)

This method returns the number of requests in the system The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

implType The implType type to use in this API call
requestType The request type to use in this API call
templateFlag True to verify if this is a template
user The user that is requesting this API call.

Returns:

int the number of requests in the system.

6.2.1.2.30 DDSAPI_Message::transferRequest

bool DDSAPI_Message::transferRequest (

std::string requestID,

DDSXML_User * fromUser,

DDSXML_User * toUser = 0

)

This method transfers the request referenced by the requestID. The caller still owns the pointer passed in. The API will not delete the pointer.

Parameters:

requestID The ID of the request to be suspended

fromUser The user to transfer the request from

toUser The user to transfer the request to

Returns:

bool True - if successful, False - if failed

6.2.1.2.31 DDSAPI_Message::getUpdateIDs

std::vector<std::string> DDSAPI_Message::getUpdateIDs (

DDSXML_Request::DDSXML_RequestTypes requestType =
DDSXML_Request::UNKNOWN_REQUEST_TYPE,

DDSXML_User * user = 0

)

This method returns all IDs of updated requests, catalogs, and templates

Parameters:

requestType The request type to use in this API call

user The user that is requesting this API call.

Returns:

vector<string> The vector containing the ID strings.

6.2.1.2.32 DDSAPI_Message::getDeletedIDs

std::vector<std::string> DDSAPI_Message::getDeletedIDs (

DDSXML_Request::DDSXML_RequestTypes requestType =
DDSXML_Request::UNKNOWN_REQUEST_TYPE,

DDSXML_User * user = 0

)

This method returns all IDs of deleted requests, catalogs, and templates

Parameters:

requestType The request type to use in this API call

user The user that is requesting this API call.

Returns:

vector<string> The vector containing the ID strings

6.2.1.2.33 DDSAPI_Message::getUsername

std::string DDSAPI_Message::getUsername (

)

This method returns the username as a string.

Returns:

std::string The username of the current user.

6.2.1.2.34 DDSAPI_Message::getVersion

std::string DDSAPI_Message::getVersion (

)

This method returns the code/XML version as a string.

Returns:

std::string The current version of the DDSAPI_Message

6.2.1.2.35 DDSAPI_Message::getClient

std::string DDSAPI_Message::getClient (

)

This method returns the clientType as a string.

Returns:

std::string The client type (i.e. API, GUI, HANDLER)

6.2.1.2.36 DDSAPI_Message::getRole

std::string DDSAPI_Message::getRole (

)

This method returns the user's role as a string.

Returns:

std::string The role that the user is currently logged in as

6.2.1.2.37 DDSAPI_Message::getDomain

std::string DDSAPI_Message::getDomain (

)

This method returns the domain of the Request Server.

Returns:

std::string The Request Server's domain.

6.2.1.2.38 DDSAPI_Message::getSubDomain

std::string DDSAPI_Message::getSubDomain (

)

This method returns the sub-domain of the Request Server.

Returns:

std::string The Request Server's sub-domain.

6.2.1.2.39 DDSAPI_Message::getUsers

DDSXML_UserList* DDSAPI_Message::getUsers (

)

This method returns the sub users of this User (if any).

Returns:

DDSXML_UserList The Sub User list.

6.2.1.2.40 DDSAPI_Message::addDestinationSetEntry

std::string DDSAPI_Message::addDestinationSetEntry (

originalDestination

destinationName

hostname

path

ftpUsername

ftpPassword

transferType

user

)

This method adds a new destination Entry to the destination Set.

Parameters:

DDSXML_Destination *originalDestination,
std::string destinationName,
std::string hostName,
std::string path,
std::string ftpUserName,
std::string ftpPassword,
DDSXML_Destination::DDSXML_DestinationTransferType transferType,
DDSXML_User* user = 0

Returns:

std::string Destination ID or "" if there was an error.

6.2.1.2.41 DDSAPI_Message::deleteDestinationSetEntry

bool DDSAPI_Message::deleteDestinationSetEntry (

destination

destinationEntryID

user

)

This method removes a user defined destination set from the user destination and if the last entry then removes the destination from the User destination list.

Parameters:

DDSXML_Destination* destination,
std::string destinationEntryID,
DDSXML_User* user = 0

Returns:

bool True if OK, False if not.

6.2.1.2.42 DDSAPI_Message::modifyDestinationSetEntry

bool DDSAPI_Message::modifyDestinationSetEntry (

originalDestination

destinationEntry

subUser)

This method modifies the Destination Entry in the Destination.

Parameters:

DDSXML_Destination *originalDestination,
DDSXML_DestinationEntry *destinationEntry,
DDSXML_User* subUser = 0

Returns:

bool True - if successful, False - if failed

6.2.1.2.43 DDSAPI_Message::moveDestinationSetEntry

bool DDSAPI_Message::moveDestinationSetEntry (

originalDestination

originalEntryID

newDestination

newEntry

subUser

)

This method moves the Destination Entry from one destination to another destination.

Parameters:

DDSXML_Destination *originalDestination,
std::string originalEntryID,
DDSXML_Destination *newDestination,
DDSXML_DestinationEntry *newEntry,
DDSXML_User *subUser = 0

Returns:

bool True - if successful, False - if failed

6.2.2 DDSXML_CatalogRequest Class Reference

This is the XML data class for the Catalog. This class is responsible for storing and maintaining the state of a catalog Request in the system. This class is not to be created outside of the API. The Class diagram representing the DDSAPI_CatalogRequest Class is provided in Figure 6.2.2-1, DDSXML_CatalogRequest UML Diagram.

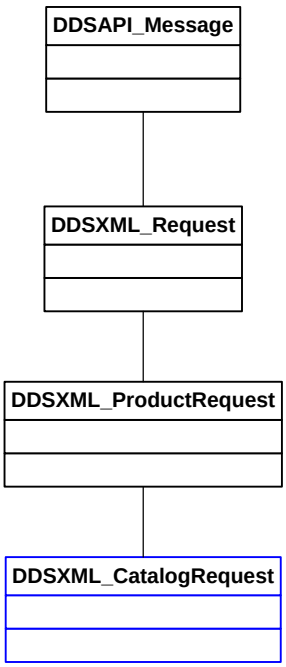


Figure 6.2.2-1, DDSXML_CatalogRequest UML Diagram

6.2.2.1 DDSXML_CatalogRequest Class Functions

6.2.2.1.1 DDSXML_CatalogRequest::getURID

std::string DDSXML_CatalogRequest::getURID (

)

Obtains the catalog's URID and returns it.

Returns:

std::string The catalog's URID as a string Empty string is not found

6.2.2.1.2 DDSXML_CatalogRequest::setURID

bool DDSXML_CatalogRequest::setURID (

std::string id

)

Set the catalog URID to query on

Parameters:

id The catalog URID to query on

Returns:

bool True if OK, else false

6.2.3 DDSXML_DataProduct Class Reference

This class is responsible for handling the data product XML. It creates, reads, writes, and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML are validated then no invalid XML should be sent across the API. Some of the method names are left to be compatible with the old API. Note that all data is now inside xml. This includes a list of data product messages.

The Class diagram representing the DDSXML_DataProduct Class is provided in Figure 6.2.3-1, DDSXML_DataProduct UML Diagram.

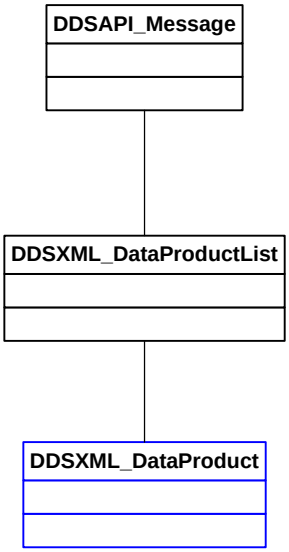


Figure 6.2.3-1, DDSXML_DataProduct UML Diagram

6.2.3.1 DDSXML_DataProduct Class Functions

6.2.3.1.1 DDSXML_DataProduct::getDataProductID

std::string DDSXML_DataProduct::getDataProductID (

)

This method returns the Data Product ID as a string. This is from the configuration file. It is a merge of the shortname and the spacecraft name.

Returns:

std::string the Data Product's ID

6.2.3.1.2 DDSXML_DataProduct::getProductShortName

std::string DDSXML_DataProduct::getProductShortName (

)

This method retrieves the Shortname. This is from the configuration file.

Returns:

std::string The Data Product's shortname.

6.2.3.1.3 DDSXML_DataProduct::getSensor

std::string DDSXML_DataProduct::getSensor (

)

This method returns the sensor for this data product as a string. This is from the configuration file.

Returns:

std::string The Data Product's sensor

6.2.3.1.4 DDSXML_DataProduct::getSpacecraft

std::string DDSXML_DataProduct::getSpacecraft (

)

This method returns the spacecraft as a string. This is from the configuration file.

For ANC data products the spacecraft will be the mission name (e.g. NPP, NPOESS), for all other data products the spacecraft will be the platform short name (e.g. NPP, N01, N02)

Returns:

std::string The spacecraft

6.2.3.1.5 DDSXML_DataProduct::getProductType

std::string DDSXML_DataProduct::getProductType (

)

This method returns the data product type as a string. This is from the configuration file.

Returns:

std::string The Data Product's type

6.2.3.1.6 DDSXML_DataProduct::getRequestType

std::string DDSXML_DataProduct::getRequestType (

)

This method returns the data product Request type as a string. This is from the configuration file.

Returns:

std::string The Data Product's request type

6.2.3.1.7 DDSXML_DataProduct::getPriority

std::string DDSXML_DataProduct::getPriority (

)

This method returns the Priority as a string. This is from the configuration file.

Returns:

std::string The Data Product's type

6.2.4 DDSXML_DataProductList Class Reference

This class is responsible for handling the data product List XML. It creates, reads, writes and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML is validated then no invalid XML should be sent across the API. This is used for the initial configuration from the Server. The request uses an ID list only.

The Class diagram representing the DDSXML_DataProductList Class is provided in Figure 6.2.4-1, DDSXML_DataProductList UML Diagram.

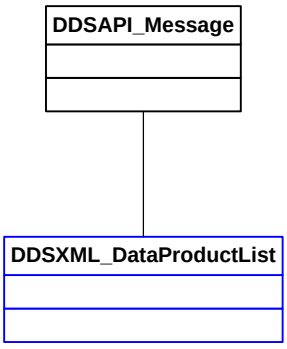


Figure 6.2.4-1, DDSXML_DataProductList UML Diagram

6.2.4.1 DDSXML_DataProductList Class Functions

6.2.4.1.1 DDSXML_DataProductList::addDataProduct

bool DDSXML_DataProductList::addDataProduct (

DDSXML_DataProduct * dataProduct

)

This method adds the data Product to this list.

Parameters:

dataProduct The data Product

Returns:

bool True if OK, else false

6.2.4.1.2 DDSXML_DataProductList::deleteDataProduct

virtual bool DDSXML_DataProductList::deleteDataProduct (

std::string dpID

)

Allows the caller to delete a data product from the list

Parameters:

dpID Deletes data product based on data product ID

Returns:

bool True if OK, else false

6.2.4.1.3 DDSXML_DataProductList::getDataProductsSet

std::set<DDSXML_DataProduct* > DDSXML_DataProductList::getDataProductsSet (

)

This method returns the DataProducts. Caller owns the data products returned.

Returns:

set<DataProduct* > The Data Product's

6.2.4.1.4 DDSXML_DataProductList::getDataProducts

std::vector<DDSXML_DataProduct* > DDSXML_DataProductList::getDataProducts (

)

This method returns the DataProducts. Caller owns the data products returned.

Returns:

vector<DataProduct* > The Data Product's

6.2.4.1.5 DDSXML_DataProductList::getDataProductList

DDSXML_DataProductList* DDSXML_DataProductList::getDataProductList (

DDSXML_DataProductIDList * dataProductIDList

)

This method returns the DataProducts for the DataProduct ID's passed in. Caller owns the data products returned.

Parameters:

dataProductIDList The list of Data product ID's to get Data products for.

Returns:

DDSXML_DataProductList* The Data Product's

6.2.4.1.6 DDSXML_DataProductList::getFilteredList

std::vector< DDSXML_DataProduct*> DDSXML_DataProductList::getFilteredList (

std::string shortname = "",

std::string spacecraft = "",

std::string sensor = "",

std::string productType = "",

std::string requestType = ""

)

This method returns a filtered vector of DataProduct pointers that matches the parameters provided by the user. If no data product(s) match the parameters provided, an empty vector is returned.

Parameters:

shortname The Shortname of the Data Product. The empty string represents all shortnames.

spacecraft The spacecraft. The empty string represents all spacecraft.

sensor The sensor. The empty string represents all sensors.

productType The data product productType. The empty string represents all categories.

requestType The data product requestType. The empty string represents all requestTypes.

Returns:

std::vector< DDSXML_DataProduct*> The vector of DDSXML_DataProducts that contain all the Data Products, requestable by the user, that meet the filter criteria.

6.2.4.1.7 DDSXML_DataProductList::getFilteredDataProductList

DDSXML_DataProductList* DDSXML_DataProductList::getFilteredDataProductList (

std::string shortname = "",

std::string spacecraft = "",

std::string sensor = "",

std::string requestType = "",

std::string productType = ""

)

This method returns a filtered vector of DataProduct pointers that match the parameters provided by the user. If no data product(s) match the parameters provided, an empty vector is returned.

Parameters:

shortname The Shortname of the Data Product. The empty string represents all shortnames.

spacecraft The spacecraft. The empty string represents all spacecraft.

sensor The sensor. The empty string represents all sensors.

productType The data product productType. The empty string represents all categories.

requestType The data product requestType. The empty string represents all requestTypes.

Returns:

DDSXML_DataProductList* The DDSXML_DataProductList of DDSXML_DataProducts that contain all the Data Products, requestable by the user, that meet the filter criteria.

6.2.4.1.8 DDSXML_DataProductList::getShortnames

std::vector<std::string> DDSXML_DataProductList::getShortnames (

)

This method returns a vector of Shortnames that can be used to query as a parameter in getFilteredList.

Returns:

std::vector<std::string> A vector of strings that contain all possible, requestable data product shortnames.

6.2.4.1.9 DDSXML_DataProductList::getSpacecrafts

std::vector<std::string> DDSXML_DataProductList::getSpacecrafts (

)

This method returns a vector of Spacecraft that can be used to query as a parameter in getFilteredList.

Returns:

std::vector<std::string> A vector of strings that contain all possible, requestable spacecraft.

6.2.4.1.10 DDSXML_DataProductList::getSensors

std::vector<std::string> DDSXML_DataProductList::getSensors (

)

This method returns a vector of Sensors that can be used to query as a parameter in getFilteredList.

Returns:

std::vector<std::string> A vector of strings that contain all possible, requestable categories.

6.2.4.1.11 DDSXML_DataProductList::getProductTypes

std::vector<std::string> DDSXML_DataProductList::getProductTypes (

)

This method returns a vector of Categories that can be used to query as a parameter in getFilteredList.

Returns:

std::vector<std::string> A vector of strings that contain all possible, requestable categories.

6.2.4.1.12 DDSXML_DataProductList::getRequestTypes

std::vector<std::string> DDSXML_DataProductList::getRequestTypes (

)

This method returns a vector of RequestTypes that can be used to query as a parameter in getFilteredList. Example

Returns:

std::vector<std::string> A vector of strings that contain all possible, requestable requestTypes.

6.2.4.1.13 DDSXML_DataProductList::getDataProduct

virtual DDSXML_DataProduct* DDSXML_DataProductList::getDataProduct (

std::string dataProductID

)

This method returns the dataProduct identified by the dataProduct ID if it exists in the system

Parameters:

dataProductID The dataProductID to check for

Returns:

int A pointer to the dataProduct . 0 - The dataProduct was not found. Valid pointer otherwise.

6.2.4.1.14 DDSXML_DataProductList::hasDataProductID

virtual bool DDSXML_DataProductList::hasDataProductID (

std::string dataProductID

)

This method returns true if the dataProduct ID exists in the system

Parameters:

dataProductID The dataProductID to check for

Returns:

int A pointer to the dataProduct . 0 - The dataProduct was not found. Valid pointer otherwise.

6.2.4.1.15 DDSXML_DataProductList::getNumberOfDataProducts

virtual int DDSXML_DataProductList::getNumberOfDataProducts (

)

This method returns the number of dataProducts for this user

Returns:

int The number of dataProducts for this user

6.2.5 DDSXML_DataShipment Class Reference

This class is responsible for handling the user shipment XML.

It creates, reads, writes and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML is validated then no invalid XML should be sent across the API. Some of the method names are left to be compatible with the old API. This class was(is) exposed to external users so we can not change the method names now.

The Class diagram representing the DDSXML_DataShipment Class is provided in Figure 6.2.5-1, DDSXML_DataShipment UML Diagram.

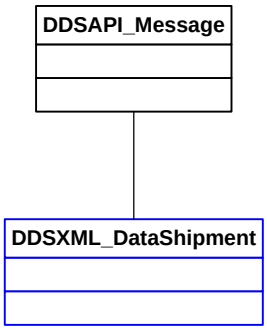


Figure 6.2.5-1, DDSXML_DataShipment UML Diagram

6.2.5.1 DDSXML_DataShipment Class Enumerations

enum DDSXML_DataShipment::DDSXML_DataShipmentStates – Valid DDS API Commands that can be sent to the Server

6.2.5.2 DDSXML_DataShipment Class Functions

6.2.5.2.1 DDSXML_DataShipment::getRequestID

std::string DDSXML_DataShipment::getRequestID (

)

This method retrieves the Request ID as a string.

Returns:

std::string The Request ID.

6.2.5.2.2 DDSXML_DataShipment::getMessage

std::string DDSXML_DataShipment::getMessage (

)

This method retrieves the Message.

Returns:

std::string The message.

6.2.5.2.3 DDSXML_DataShipment::getURID

std::string DDSXML_DataShipment::getURID (

)

This method retrieves the URID.

Returns:

std::string The URID

6.2.5.2.4 DDSXML_DataShipment::getFileName

std::string DDSXML_DataShipment::getFileName (

)

This method retrieves the FileName.

Returns:

std::string The FileName associated with this shipment.

6.2.5.2.5 DDSXML_DataShipment::getTimestamp

Int64 DDSXML_DataShipment::getTimestamp (

)

This method retrieves the Timestamp in IET (microseconds).

Returns:

Int64 The Timestamp

6.2.5.2.6 DDSXML_DataShipment::getTransferTime

Int64 DDSXML_DataShipment::getTransferTime (

)

This method retrieves the TransferTime in IET (microseconds).

Returns:

Int64 The Transfer Time

6.2.5.2.7 DDSXML_DataShipment::getDataShipmentState

DDSXML_DataShipment::DDSXML_DataShipmentStates

DDSXML_DataShipment::getDataShipmentState (

)

This method retrieves the last state of this shipment

Returns:

DDSXML_DataShipmentStates The state associated with this shipment

6.2.5.2.8 DDSXML_DataShipment::getFTPUserName

std::string DDSXML_DataShipment::getFTPUserName (

)

This method retrieves the FTP User Name.

Returns:

std::string The FTPUserName.

6.2.5.2.9 DDSXML_DataShipment::getCollectionShortName

std::string DDSXML_DataShipment::getCollectionShortName (

)

This method retrieves the Collection Short Name.

Returns:

std::string The Collection Short Name.

6.2.5.2.10 Deleted

6.2.5.2.11 DDSXML_DataShipment::getHostName

std::string DDSXML_DataShipment::getHostName (

)

This method retrieves the Host Name or IP address.

Returns:

std::string The HostName.

6.2.5.2.12 DDSXML_DataShipment::getFilePath

std::string DDSXML_DataShipment::getFilePath (

)

This method retrieves the FilePath.

Returns:

std::string The FilePath.

6.2.5.2.13 DDSXML_DataShipment::getCheckSum

std::string DDSXML_DataShipment::getCheckSum (

)

This method retrieves the CheckSum of the HDF5 file shipped.

Returns:

std::string The CheckSum.

6.2.5.2.14 DDSXML_DataShipment::getFileSize

int DDSXML_DataShipment::getFileSize (

)

This method retrieves the File Size.

Returns:

std::string The FileSize

6.2.5.2.15 DDSXML_DataShipment::getDestinationID

std::string DDSXML_DataShipment::getDestinationID (

)

This method retrieves the Destination ID.

Returns:

std::string The Destination ID.

6.2.5.2.16 DDSXML_DataShipment::getDestinationName

std::string DDSXML_DataShipment::getDestinationName (

)

This method retrieves the Destination Name.

Returns:

std::string The Destination Name.

6.2.5.2.17 DDSXML_DataShipment::getMasterDestinationIndex

std::string DDSXML_DataShipment::getMasterDestinationIndex (

)

This method retrieves the Master Destination Index.

Returns:

std::string The Master Destination Index.

6.2.6 DDSXML_Destination Class Reference

This class is responsible for handling the user destination XML.

It creates, reads, writes and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML is validated then no invalid XML should be sent across the API. Some of the method names are left to be compatible with the old API. This class was(is) exposed to external users so we can not change the method names now.

The Class diagram representing the DDSXML_Destination Class is provided in Figure 6.2.6-1, DDSXML_Destination UML Diagram.

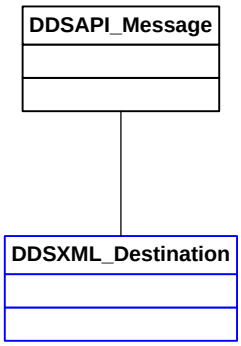


Figure 6.2.6-1, DDSXML_Destination UML Diagram

6.2.6.1 DDSAPI_Message Class Attributes

std::string DDSXML_Destination::DDSXML_DestinationStatesName[4] – DDS API Client type names

DDSXML_Destination::DDSXML_DestinationTransferTypeName[5] – DDS API Client type names

6.2.6.2 DDSAPI_Message Class Enumerations

enum DDSXML_Destination::DDSXML_DestinationStates – Valid DDS API Commands that can be sent to the Server

DDSXML_Destination::DDSXML_DestinationTransferType – Valid DDS API Commands that can be sent to the Server

6.2.6.3 DDSAPI_Message Class Functions

6.2.6.3.1 DDSXML_Destination::getDestinationName

std::string DDSXML_Destination::getDestinationName (

)

This method retrieves the Destination Name as a string.

Returns:

std::string The destination Name or "" if not valid.

6.2.6.3.2 DDSXML_Destination::getUserPassword

std::string DDSXML_Destination::getUserPassword (

)

This method retrieves the myPassword attribute's value as a string. The value returned will be clear text.

Returns:

std::string The password as a string or "" if not valid.

6.2.6.3.3 DDSXML_Destination::getPath

std::string DDSXML_Destination::getPath (

)

This method retrieves the myPath attribute's value as a string.

Returns:

std::string The destination path as a string or "" if not valid.

6.2.6.3.4 DDSXML_Destination::getHostName

std::string DDSXML_Destination::getHostName (

)

This method returns the destination host name (or IP) Note that this is really a valid hostname or IP address that can be used for FTP.

Returns:

std::string The Host name as a string or "" if not valid.

6.2.6.3.5 DDSXML_Destination::getUserName

std::string DDSXML_Destination::getUserName (

)

This method retrieves the myUsername attribute's value as a string.

Returns:

std::string The username associated with this destination or "" if not valid.

6.2.6.3.6 DDSXML_Destination::getOwner

DDSXML_User* DDSXML_Destination::getOwner (

)

This method retrieves the owner of this destination

Returns:

DDSXML_User The owner of this destination or NULL if not valid.

6.2.6.3.7 DDSXML_Destination::getState

DDSXML_Destination::DDSXML_DestinationStates DDSXML_Destination::getState (

)

This method retrieves the last state of this destination

Returns:

DDSXML_DestinationStates The state associated with this destination

6.2.6.3.8 DDSXML_Destination::getStateName

std::string DDSXML_Destination::getStateName (

)

This method retrieves the last state of this destination as a string

Returns:

std::string The state name as a string

6.2.6.3.9 DDSXML_Destination::getTransferType

DDSXML_Destination::DDSXML_DestinationTransferType
DDSXML_Destination::getTransferType (

)

This method retrieves the transfer Type of this destination

Returns:

DDSXML_DestinationTransferType The transfer Type

6.2.6.3.10 DDSXML_Destination::getTransferTypeName

std::string DDSXML_Destination::getTransferTypeName (

)

This method retrieves the transfer Type name.

Returns:

std::string The transfer Type name

6.2.6.3.11 DDSXML_Destination::setDestinationName

bool DDSXML_Destination::setDestinationName (

const std::string & destinationName

)

This method sets the name of the destination. This is a user defined name.

Parameters:

destinationName The name used to set destination name.

Returns:

bool True if set OK, False if not.

6.2.6.3.12 DDSXML_Destination::setUserPassword

bool DDSXML_Destination::setUserPassword (

const std::string & password

)

This method sets the FTP Password.

Parameters:

password The password.

Returns:

bool True if set OK, False if not.

6.2.6.3.13 DDSXML_Destination::setPath

bool DDSXML_Destination::setPath (

const std::string & path

)

This method sets the Path.

Parameters:

path The path.

Returns:

bool True if set OK, False if not.

6.2.6.3.14 DDSXML_Destination::setHostName

bool DDSXML_Destination::setHostName (

const std::string & hostName

)

This method sets the destination host name Note that this is really a valid hostname or IP address that can be used for FTP.

Parameters:

hostName The host name.

Returns:

bool True if set OK, False if not.

6.2.6.3.15 DDSXML_Destination::setUserName

bool DDSXML_Destination::setUserName (

const std::string & userName

)

This method sets the FTP user.

Parameters:

userName The user name.

Returns:

bool True if set OK, False if not.

6.2.6.3.16 DDSXML_Destination::setOwner

bool DDSXML_Destination::setOwner (

DDSXML_User * owner

)

This method sets The owner of this destination

Parameters:

owner The owner of this destination.

Returns:

bool True if set OK, False if not.

6.2.6.3.17 DDSXML_Destination::setState

bool DDSXML_Destination::setState (

DDSXML_Destination::DDSXML_DestinationStates state

)

This method sets the state attribute in the Destination class.

Parameters:

state The state of the Destination.

Returns:

bool True if set OK, False if not.

6.2.6.3.18 DDSXML_Destination::setTransferType

bool DDSXML_Destination::setTransferType (

DDSXML_Destination::DDSXML_DestinationTransferType transferType

)

This method sets the transfer Type attribute in the Destination class. This will allow the destination to be sent using different tranfer methods.

Parameters:

transferType The transfer Type of the Destination.

Returns:

bool True if set OK, False if not.

6.2.6.3.19 DDSXML_Destination::getUserIndex

std::string DDSXML_Destination::getUserIndex (

)

This method returns the user Destination index

Returns:

std::string The user destination Index as a string or "" if not valid.

6.2.6.3.20 DDSXML_Destination::getMasterIndex

std::string DDSXML_Destination::getMasterIndex (

)

This method returns the master Destination index

Returns:

std::string The Master Index as a string or "" if not valid.

6.2.6.3.21 DDSXML_Destination::getDestinationEntryName

std::string DDSXML_Destination::getDestinationEntryName (

)

This method retrieves the Destination Name as a string

Returns:

std::string The destination Name or "" if not valid.

6.2.6.3.22 DDSXML_Destination::getFTPUserPassword

std::string DDSXML_Destination::getFTPUserPassword (

)

This method retrieves the FTP Password attribute's value as a string. The value returned will be cleartext

Returns:

std::string The FTP password as a string or "" if not valid.

6.2.6.3.23 DDSXML_Destination::getPath

std::string DDSXML_Destination::getPath (

)

This method retrieves the myPath attribute's value as a string.

Returns:

std::string The destination path as a string or "" if not valid.

6.2.6.3.24 DDSXML_Destination::getHostName

std::string DDSXML_Destination::getHostName (

)

This method returns the destination host name (or IP) Note that this is really a valid hostname or IP address that can be used for FTP.

Returns:

std::string The Host name as a string or "" if not valid.

6.2.6.3.25 DDSXML_Destination::getFTPUserName

std::string DDSXML_Destination::getFTPUserName (

)

This method retrieves the FTP Username attribute's value as a string.

Returns:

std::string The FTP username associated with this destination or "" if not valid.

6.2.6.3.26 DDSXML_Destination::getOwner

DDSXML_User DDSXML_Destination::getOwner (

)

This method retrieves the owner of this destinationEntry.

Returns:

DDSXML_User The owner of this destinationEntry or NULL if not valid.

6.2.6.3.27 DDSXML_Destination::getState

DDSXML_Destination::DDSXML_DestinationStates DDSXML_Destination::getState (

)

This method retrieves the last state of this destinationEntry.

Returns:

DDSXML_DestinationEntryStates The state associated with this destinationEntry

6.2.6.3.28 DDSXML_Destination::getStateName

std::string DDSXML_Destination::getStateName (

)

This method retrieves the last state of this destinationEntry as a string.

Returns:

std::string The state name as a string.

6.2.6.3.29 DDSXML_Destination::getTransferType

DDSXML_Destination::DDSXML_DestinationTransferType
DDSXML_Destination::getTransferType (

)

This method retrieves the transfer Type of this destinationEntry.

Returns:

DDSXML_DestinationTransferType The transfer Type.

6.2.6.3.30 DDSXML_Destination::getTransferTypeName

std::string DDSXML_Destination::getTransferTypeName (

)

This method retrieves the transfer Type name.

Returns:

std::string The transfer Type name.

6.2.6.3.31 DDSXML_Destination::setDestinationEntryName

bool DDSXML_Destination::setDestinationEntryName (

destinationName

)

This method sets the name of the destination. This is a user defined name.

Parameters:

const std::string& destinationName.

Returns:

bool True if set OK, False if not.

6.2.6.3.32 DDSXML_Destination::setFTPUserPassword

bool DDSXML_Destination::setFTPUserPassword (

password

)

This method sets the FTP Password.

Parameters:

const std::string& password.

Returns:

bool True if set OK, False if not.

6.2.6.3.33 DDSXML_Destination::setPath

bool DDSXML_Destination::setPath (

path

)

This method sets the Path.

Parameters:

const std::string& path

Returns:

bool True if set OK, False if not.

6.2.6.3.34 DDSXML_Destination::setHostName

bool DDSXML_Destination::setHostName (

hostName

)

This method sets the destination host name

Note that this is really a valid hostname or IP address that can be used for FTP.

Parameters:

const std::string& hostName

Returns:

bool True if set OK, False if not.

6.2.6.3.35 DDSXML_Destination::setFTPUserName

bool DDSXML_Destination::setFTPUserName (

userName

)

This method sets the FTP user.

Parameters:

const std::string& userName

Returns:

bool True if set OK, False if not.

6.2.6.3.36 DDSXML_Destination::setOwner

bool DDSXML_Destination::setOwner (

owner

)

This method sets the owner of this destination.

Parameters:

DDSXML_User* owner

Returns:

bool True if set OK, False if not.

6.2.6.3.37 DDSXML_Destination::setState

bool DDSXML_Destination::setState (

state

)

This method sets the state attribute in the Destination class.

Parameters:

DDSXML_Destination::DDSXML_DestinationStates state

Returns:

bool True if set OK, False if not.

6.2.6.3.38 DDSXML_Destination::setTransferType

bool DDSXML_Destination::setTransferType (

transferType

)

This method sets the transfer Type attribute in the Destination class. This will allow the destinationEntry to be sent using different transfer methods.

.**Parameters:**

DDSXML_Destination::DDSXML_DestinationTransferType transferType

Returns:

bool True if set OK, False if not.

6.2.6.3.39 DDSXML_Destination::getEntryIndex

std::string DDSXML_Destination::getEntryIndex (

)

This method returns the user Destination Entry index.

.

Returns:

string The user destination Entry Index as a string or "" if not valid.

6.2.7 DDSXML_GEORequest Class Reference

This is the XML data class for the GEO Request. This class is responsible for storing and maintaining the state of a request in the system.

The Class diagram representing the DDSXML_GEORequest Class is provided in Figure 6.2.7-1, DDSXML_GEORequest UML Diagram.

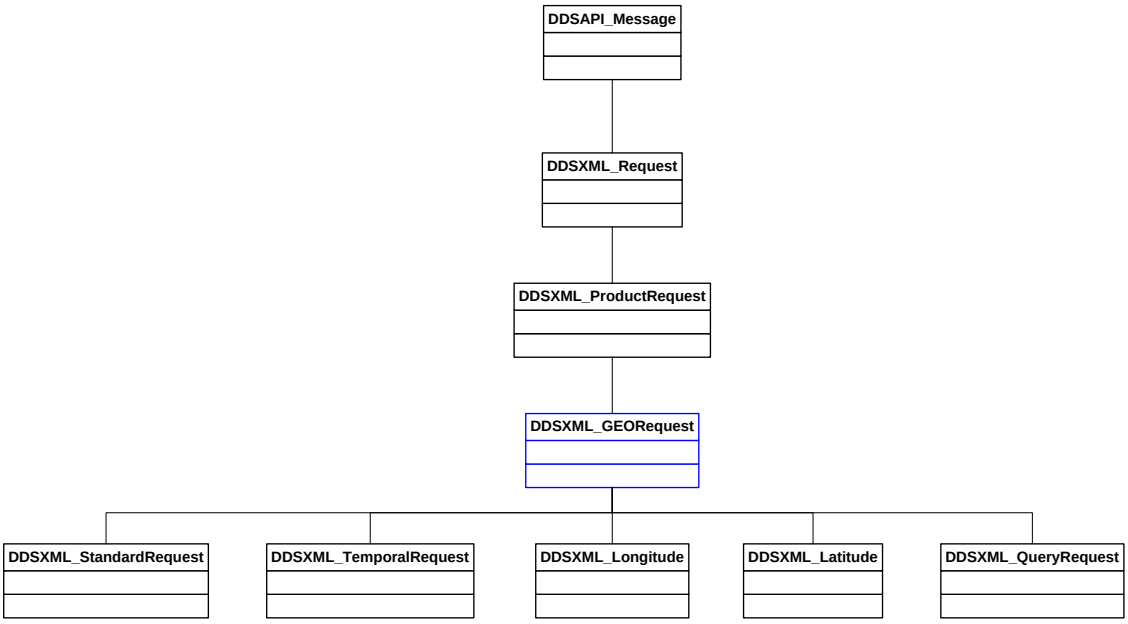


Figure 6.2.7-1, DDSXML_GEORequest UML Diagram

6.2.7.1 DDSXML_GEORequest Class Functions

6.2.7.1.1 DDSXML_GEORequest::getAggInterval

Int64 DDSXML_GEORequest::getAggInterval (

)

This method returns the value of AggInterval. The time is in IET (microseconds). getAggIntervalUsedFlag() must return true for this field to be valid. getAggregationEnabled() must return true for this field to be valid.

Returns:

Int64 The aggregation interval.

6.2.7.1.2 DDSXML_GEORequest::getDelay

Int64 DDSXML_GEORequest::getDelay (

)

This method returns the delay for the request in IET. The time is in IET (microseconds).

Returns:

Int64 The request's processing delay

6.2.7.1.3 DDSXML_GEORequest::getLowerRightLatitude

DDSXML_Latitude* DDSXML_GEORequest::getLowerRightLatitude (

)

This method returns the Lower Right Latitude. getGeospatialUsedFlag() must return true for this field to be valid. isGeospatial() must return true for this field to be valid. getLowerRightLatitudeUsedFlag() must return true for this field to be valid. This memory is owned by the caller and should be freed/deleted by the caller.

Returns:

DDSXML_Latitude* A pointer to a DDSXML_Latitude* object that contains the lower right latitude.

6.2.7.1.4 DDSXML_GEORequest::getLowerRightLongitude

DDSXML_Longitude* DDSXML_GEORequest::getLowerRightLongitude (

)

This method returns the Lower Right Longitude. getGeospatialUsedFlag() must return true for this field to be valid. isGeospatial() must return true for this field to be valid. getLowerRightLongitudeUsedFlag() must return true for this field to be valid. This memory is owned by the caller and should be freed/deleted by the caller.

Returns:

DDSXML_Longitude* A pointer to a DDSXML_Longitude* object that contains the lower right longitude.

6.2.7.1.5 DDSXML_GEORequest::getUpperLeftLatitude

DDSXML_Latitude* DDSXML_GEORequest::getUpperLeftLatitude (

)

This method returns the Upper Left Latitude. getGeospatialUsedFlag() must return true for this field to be valid. isGeospatial() must return true for this field to be valid. getUpperLeftLatitudeUsedFlag() must return true for this field to be valid. This memory is owned by the caller and should be freed/deleted by the caller.

Returns:

DDSXML_Latitude* A pointer to a DDSXML_Latitude* object that contains the upper left latitude.

6.2.7.1.6 DDSXML_GEORequest::getUpperLeftLongitude

DDSXML_Longitude* DDSXML_GEORequest::getUpperLeftLongitude (

)

This method returns the Upper Left Longitude. getGeospatialUsedFlag() must return true for this field to be valid. isGeospatial() must return true for this field to be valid. getUpperLeftLongitudeUsedFlag() must return true for this field to be valid. This memory is owned by the caller and should be freed/deleted by the caller.

Returns:

DDSXML_Longitude* A pointer to a DDSXML_Longitude* object that contains the upper left longitude.

6.2.7.1.7 DDSXML_GEORequest::getStartOrbitRevolution

int DDSXML_GEORequest::getStartOrbitRevolution (

)

This method returns the starting orbit revolution for the request. This must be between getMinimumOrbitRevolution() and getMaximumOrbitRevolution(); getStartOrbitRevolutionUsedFlag() must return true for this field to be valid. getOrbitIDEnabled() must return true for this field to be valid.

Returns:

int The orbit revolution number.

6.2.7.1.8 DDSXML_GEORequest::getEndOrbitRevolution

int DDSXML_GEORequest::getEndOrbitRevolution (

)

This method returns the last orbit revolution for the request. This must be between getMinimumOrbitRevolution() and getMaximumOrbitRevolution(); getStartOrbitRevolutionUsedFlag() must return true for this field to be valid. getOrbitIDEnabled() must return true for this field to be valid.

Returns:

int The orbit revolution number.

6.2.7.1.9 DDSXML_GEORequest::getRepaired

bool DDSXML_GEORequest::getRepaired (

)

This method returns the Repaired flag If true then repaired data will be sent for this request. getRepairedUsedFlag() must return true for this field to be valid.

Returns:

bool The repaired status flag.

6.2.7.1.10 DDSXML_GEORequest::getPackageState

bool DDSXML_GEORequest::getPackageState (

)

This method returns the package state of the request. If true then data will be packaged for this request. getPackageStateUsedFlag() must return true for this field to be valid.

Returns:

bool The package state of the request.

6.2.7.1.11 DDSXML_GEORequest::isGeospatial

bool DDSXML_GEORequest::isGeospatial (

)

This method returns the status of the geospatial subset. If this is true all four Latitude and Longitude elements must be filled in. getGeospatialUsedFlag() must return true for this field to be valid.

Returns:

bool true = The geospatial subset is on false = The geospatial subset is off

6.2.7.1.12 DDSXML_GEORequest::setAggInterval

bool DDSXML_GEORequest::setAggInterval (

Int64 aggInterval

)

This method sets the AggInterval. The time is in IET (microseconds). This must be between getMinimumAggregationInterval() and getMaximumAggregationInterval(); getAggIntervalUsedFlag() must return true for this field to be valid. setAggregationEnabled() must be set to true for this field to be valid.

Parameters:

aggInterval The aggregation interval.

Returns:

bool - True if OK, else false

6.2.7.1.13 DDSXML_GEORequest::setDelay

bool DDSXML_GEORequest::setDelay (

Int64 delay

)

This method sets the delay. The time is in IET (microseconds). This must be between getMinimumDelay() and getMaximumDelay(); getDelayUsedFlag() must return true for this field to be valid. If set to 0 there is no delay.

Parameters:

delay The delay interval.

Returns:

bool - True if OK, else false

6.2.7.1.14 DDSXML_GEORequest::setGeospatial

bool DDSXML_GEORequest::setGeospatial (

bool geospatialFlag

)

This method sets the geospatialFlag. If this is set all four Latitude and Longitude elements must be filled in. getGeospatialUsedFlag() must return true for this field to be valid.

Parameters:

geospatialFlag is the GeoSpatialSubset active

Returns:

bool - True if OK, else false

6.2.7.1.15 DDSXML_GEORequest::setLowerRightLatitude

bool DDSXML_GEORequest::setLowerRightLatitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Lower Right Latitude. getGeospatialUsedFlag() must return true for this field to be valid. setGeospatial() must be set to true for this field to be valid. getLowerRightLatitudeUsedFlag() must return true for this field to be valid. This method will take a Latitude value that is $-90 \leq deg \leq 90$.

Parameters:

isNegative Specifies if the LatLong is negative
degrees The degrees value for the latitude or longitude as an integer.
minutes The minutes value for the latitude or longitude as an integer.

seconds The seconds value for the latitude or longitude as a float.

Returns:

- bool - True if OK, else false

6.2.7.1.16 DDSXML_GEORequest::setLowerRightLongitude

bool DDSXML_GEORequest::setLowerRightLongitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Lower Right Longitude getGeospatialUsedFlag() must return true for this field to be valid. setGeospatial() must be set to true for this field to be valid. getLowerRightLongitudeUsedFlag() must return true for this field to be valid. This method will take a Lat Long value that is 179 < deg < 360 and subtract 360 degrees from it, to put it in the -180 to +179 range. This will make this lat long value go from a 0 to 359 positioning system to the DDS server standard system of -180 to +179 (where zero in both systems are the same place).

Parameters:

- isNegative Specifies if the LatLong is negative
- degrees The degrees value for the latitude or longitude as an integer.
- minutes The minutes value for the latitude or longitude as an integer.
- seconds The seconds value for the latitude or longitude as a float.

Returns:

- bool - True if OK, else false

6.2.7.1.17 DDSXML_GEORequest::setStartOrbitRevolution

bool DDSXML_GEORequest::setStartOrbitRevolution (

int orbitRevolution

)

This method sets the start orbit revolution for the Request. This must be between getMinimumOrbitRevolution() and getMaximumOrbitRevolution(); getStartOrbitRevolutionUsedFlag() must return true for this field to be valid. getOrbitIDEnabled() must return true for this field to be valid.

Parameters:

- orbitRevolution The new orbit revolution for the request.

Returns:

- bool - True if OK, else false

6.2.7.1.18 DDSXML_GEORequest::setUpperLeftLatitude

bool DDSXML_GEORequest::setUpperLeftLatitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Upper Left Latitude getGeospatialUsedFlag() must return true for this field to be valid. setGeospatial() must be set to true for this field to be valid. getUpperLeftLatitudeUsedFlag() must return true for this field to be valid. This method will take a Latitude value that is $-90 \leq deg \leq 90$.

Parameters:

- isNegative Specifies if the LatLong is negative
- degrees The degrees value for the latitude or longitude as an integer.
- minutes The minutes value for the latitude or longitude as an integer.
- seconds The seconds value for the latitude or longitude as a float.

Returns:

- bool - True if OK, else false

6.2.7.1.19 DDSXML_GEORequest::setUpperLeftLongitude

bool DDSXML_GEORequest::setUpperLeftLongitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Upper Left Longitude getGeospatialUsedFlag() must return true for this field to be valid. setGeospatial() must be set to true for this field to be valid. getUpperLeftLongitudeUsedFlag() must return true for this field to be valid. This method will take a Lat Long value that is 179 < deg < 360 and subtract 360 degrees from it, to put it in the -180 to +179 range. This will make this lat long value go from a 0 to 359 positioning system to the DDS server standard system of -180 to +179 (where zero in both systems are the same place).

Parameters:

- isNegative Specifies if the LatLong is negative
- degrees The degrees value for the latitude or longitude as an integer.
- minutes The minutes value for the latitude or longitude as an integer.
- seconds The seconds value for the latitude or longitude as a float.

Returns:

- bool - True if OK, else false

6.2.7.1.20 DDSXML_GEORequest::setEndOrbitRevolution

bool DDSXML_GEORequest::setEndOrbitRevolution (

int orbitRevolution

)

This method sets the end orbit revolution for the Request. This must be between getMinimumOrbitRevolution() and getMaximumOrbitRevolution(); getStartOrbitRevolutionUsedFlag() must return true for this field to be valid. getOrbitIDEnabled() must return true for this field to be valid.

Parameters:

- orbitRevolution The new orbit revolution for the request.

Returns:

- bool - True if OK, else false

6.2.7.1.21 DDSXML_GEORequest::setRepaired

bool DDSXML_GEORequest::setRepaired (

bool repaired

)

This method sets the Repaired flag

Parameters:

- repaired The new repaired state for the request.

Returns:

- bool - True if OK, else false

6.2.7.1.22 DDSXML_GEORequest::setPackageState

bool DDSXML_GEORequest::setPackageState (

bool state

)

Sets the state of each package in the request to the value passed in.

Parameters:

- state - The state of the package.

Returns:

- bool - True if OK, else false

6.2.7.1.23 DDSXML_GEORequest::getAggIntervalUsedFlag

bool DDSXML_GEORequest::getAggIntervalUsedFlag (

)

This method returns the value of the used flag True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.24 DDSXML_GEORequest::getDelayUsedFlag

bool DDSXML_GEORequest::getDelayUsedFlag (

)

This method returns the value of the used flag True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.25 DDSXML_GEORequest::getGeospatialUsedFlag

bool DDSXML_GEORequest::getGeospatialUsedFlag (

)

This method returns the value of the used flag True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.26 DDSXML_GEORequest::getLowerRightLatitudeUsedFlag

bool DDSXML_GEORequest::getLowerRightLatitudeUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type. Caller owns the object returned

Returns:

- bool True if used in this request type.

6.2.7.1.27 DDSXML_GEORequest::getLowerRightLongitudeUsedFlag

bool DDSXML_GEORequest::getLowerRightLongitudeUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.28 DDSXML_GEORequest::getUpperLeftLatitudeUsedFlag

bool DDSXML_GEORequest::getUpperLeftLatitudeUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.29 DDSXML_GEORequest::getUpperLeftLongitudeUsedFlag

bool DDSXML_GEORequest::getUpperLeftLongitudeUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.30 DDSXML_GEORequest::getStartOrbitRevolutionUsedFlag

bool DDSXML_GEORequest::getStartOrbitRevolutionUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.31 DDSXML_GEORequest::getEndOrbitRevolutionUsedFlag

bool DDSXML_GEORequest::getEndOrbitRevolutionUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.32 DDSXML_GEORequest::getRepairedUsedFlag

bool DDSXML_GEORequest::getRepairedUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.33 DDSXML_GEORequest::getPackageStateUsedFlag

bool DDSXML_GEORequest::getPackageStateUsedFlag (

)

This method returns the value of the used flag. True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.7.1.34 DDSXML_GEORequest::getOrbitIDEnabled

bool DDSXML_GEORequest::getOrbitIDEnabled (

)

This method returns the value of the flag

Returns:

- bool True if enabled in this request.

6.2.7.1.35 DDSXML_GEORequest::setOrbitIDEnabled

bool DDSXML_GEORequest::setOrbitIDEnabled (

bool flag

)

This method sets the value of the flag

Parameters:

- flag True if this is enabled

Returns:

- bool - True if OK, else false

6.2.7.1.36 DDSXML_GEORequest::getAggregationEnabled

bool DDSXML_GEORequest::getAggregationEnabled (

)

This method returns the value of the flag

Returns:

- bool True if enabled in this request.

6.2.7.1.37 DDSXML_GEORequest::setAggregationEnabled

bool DDSXML_GEORequest::setAggregationEnabled (

bool flag

)

This method sets the value of the flag

Parameters:

- flag True if this is enabled

Returns:

- bool - True if OK, else false

6.2.7.1.38 DDSXML_GEORequest::getMinimumDelay

Int64 DDSXML_GEORequest::getMinimumDelay (

)

This method returns the MinimumDelay. This is in microseconds.

Returns:

- Int64 The MinimumDelay

6.2.7.1.39 DDSXML_GEORequest::getMaximumDelay

Int64 DDSXML_GEORequest::getMaximumDelay (

)

This method returns the MaximumDelay. This is in microseconds.

Returns:

- Int64 The MaximumDelay

6.2.7.1.40 DDSXML_GEORequest::getMinimumAggregationInterval

Int64 DDSXML_GEORequest::getMinimumAggregationInterval (

)

This method returns the MinimumAggregationInterval. This is in microseconds.

Returns:

- Int64 The MinimumAggregationInterval

6.2.7.1.41 DDSXML_GEORequest::getMaximumAggregationInterval

Int64 DDSXML_GEORequest::getMaximumAggregationInterval (

)

This method returns the MaximumAggregationInterval. This is in microseconds.

Returns:

- Int64 The MaximumAggregationInterval

6.2.8 DDSXML_PeriodicRequest Class Reference

This is the XML data class for the Periodic Request. This class is responsible for storing and maintaining the state of a request in the system.

The Class diagram representing the DDSXML_PeriodicRequest Class is provided in Figure 6.2.8-1, DDSXML_PeriodicRequest UML Diagram.

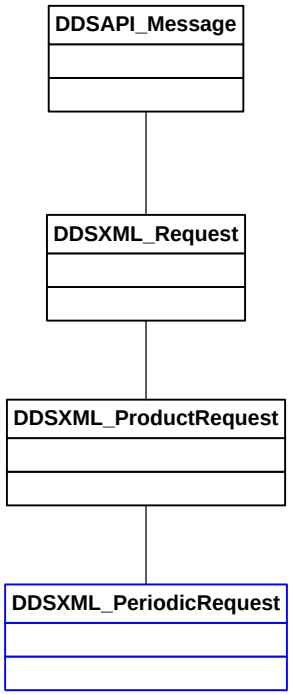


Figure 6.2.8-1, DDSXML_PeriodicRequest UML Diagram

6.2.8.1 DDSXML_PeriodicRequest Class Functions

6.2.8.1.1 DDSXML_PeriodicRequest::getDays

int DDSXML_PeriodicRequest::getDays (

)

This method returns the value of the days as an int

Returns:

- int The days for the request.

6.2.8.1.2 DDSXML_PeriodicRequest::setDays

bool DDSXML_PeriodicRequest::setDays (

int days

)

This method sets the days to the value specified by the parameter.

Parameters:

- days The days for the request.

Returns:

- bool - True if Ok, else error

6.2.8.1.3 DDSXML_PeriodicRequest::getHours

int DDSXML_PeriodicRequest::getHours (

)

This method returns the value of the Hours as an int

Returns:

- int The Hours for the request.

6.2.8.1.4 DDSXML_PeriodicRequest::setHours

bool DDSXML_PeriodicRequest::setHours (

int hours

)

This method sets the Hours to the value specified by the parameter.

Parameters:

- hours The Hours for the request.

Returns:

- bool - True if Ok, else error

6.2.8.1.5 DDSXML_PeriodicRequest::getMinutes

int DDSXML_PeriodicRequest::getMinutes (

)

This method returns the value of the Minutes as an int

Returns:

- int The Minutes for the request.

6.2.8.1.6 DDSXML_PeriodicRequest::setMinutes

bool DDSXML_PeriodicRequest::setMinutes (

int minutes

)

This method sets the Minutes to the value specified by the parameter.

Parameters:

- minutes The days for the request.

Returns:

- bool - True if Ok, else error

6.2.8.1.7 DDSXML_PeriodicRequest::getSeconds

int DDSXML_PeriodicRequest::getSeconds (

)

This method returns the value of the Seconds as an int

Returns:

- int The Seconds for the request.

6.2.8.1.8 DDSXML_PeriodicRequest::setSeconds

bool DDSXML_PeriodicRequest::setSeconds (

int seconds

)

This method sets the Seconds to the value specified by the parameter.

Parameters:

- seconds The Seconds for the request.

Returns:

- bool - True if Ok, else error

6.2.8.1.9 DDSXML_PeriodicRequest::getPeriodicityUsedFlag

bool DDSXML_PeriodicRequest::getPeriodicityUsedFlag (

)

This method returns the value of the used flag True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.8.1.10 DDSXML_PeriodicRequest::getPeriodicityEnabled

bool DDSXML_PeriodicRequest::getPeriodicityEnabled (

)

This method returns the value of the enabled flag True is returned if the value is enabled in this request type.

Returns:

- bool True if enabled in this request type.

6.2.8.1.11 DDSXML_PeriodicRequest::setPeriodicityEnabled

bool DDSXML_PeriodicRequest::setPeriodicityEnabled (

bool flag

)

This method sets the value of the enabled flag True is enabled in this request type.

Parameters:

- flag True if enabled for this request.

Returns:

- bool True if set, else false.

6.2.9 DDSXML_ProductRequest Class Reference

This is the XML data class for the Product Request. This class is responsible for storing and maintaining the state of a request in the system.

The Class diagram representing the DDSXML_ProductRequest Class is provided in Figure 6.2.9-1, DDSXML_ProductRequest UML Diagram.

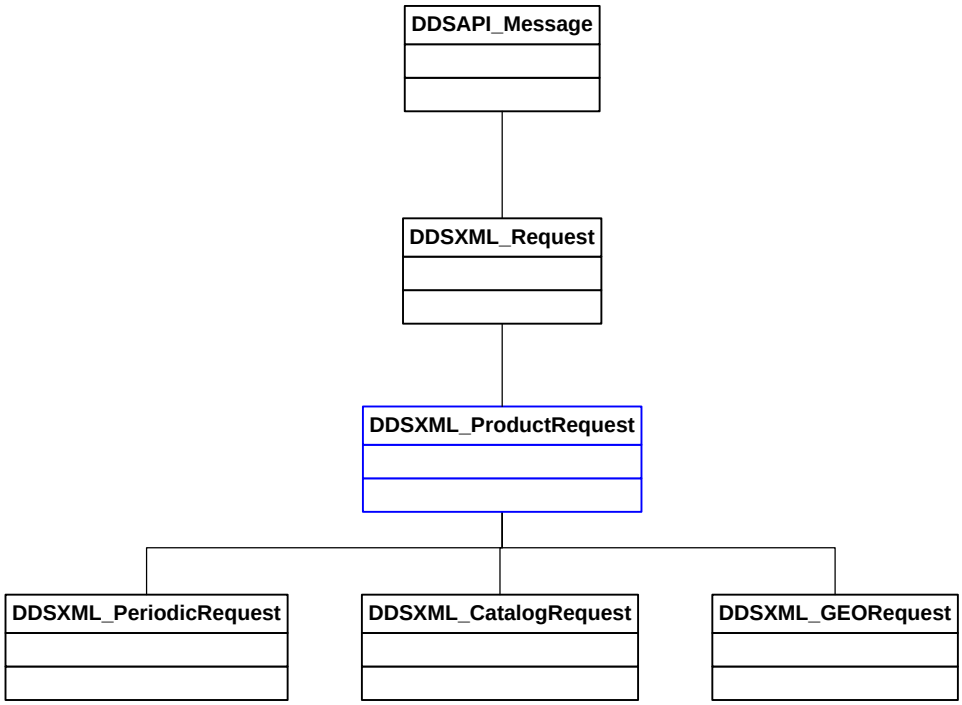


Figure 6.2.9-1, DDSXML_ProductRequest UML Diagram

6.2.10 DDSXML_QueryRequest Class Reference

This is the XML data class for the Query Request. This class is responsible for storing and maintaining the state of a request in the system.

The Class diagram representing the DDSXML_QueryRequest Class is provided in Figure 6.2.10-1, DDSXML_QueryRequest UML Diagram.

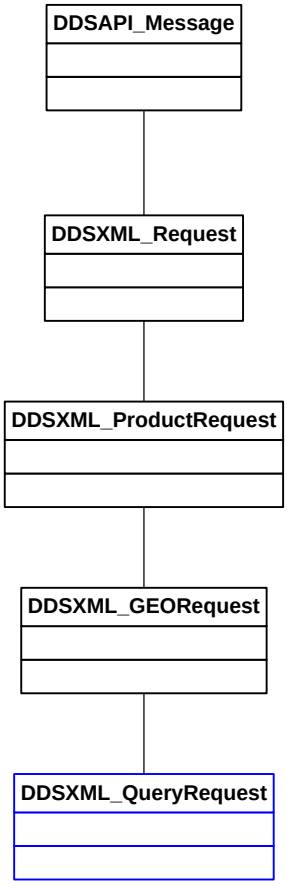


Figure 6.2.10-1, DDSXML_QueryRequest UML Diagram

6.2.11 DDSXML_Request Class Reference

This is the base class for all Request types. This class is responsible for storing and maintaining the base state of a request in the system. A Request is made up of a product request type and an implementation request type. Based on the selection made at creation time a specific request will be created and returned. The request returned will be one of the implementation types. This will allow for the validation of the request and The ability to correctly define the progress object. The Data ID (Old Request ID, States, User etc are in the base DDSXML_UserData class. This allows all data to be treated by the API, Server, and Handler etc using the same methods for information common to all. This does not include the progress of the Request. (Internal use only) The state of the request can be viewed from the base DDSXML_UserData Class. This does not include the messages of the request.

The Class diagram representing the DDSXML_Request Class is provided in Figure 6.2.11-1, DDSXML_Request UML Diagram.

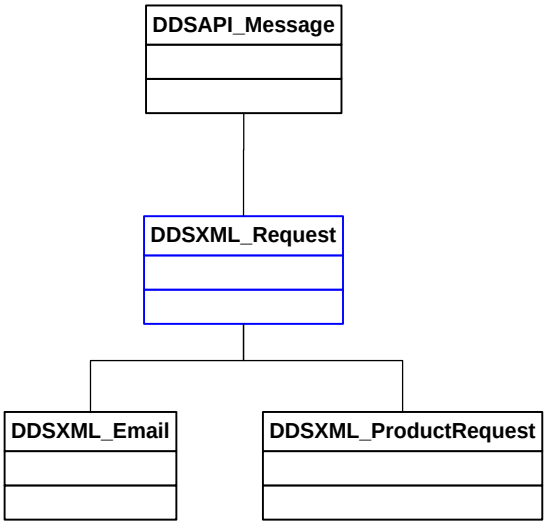


Figure 6.2.11-1, DDSXML_Request UML Diagram

6.2.11.1 DDSAPI_Message Class Attributes

- std::string DDSXML_Request::DDSXML_RequestTypesNames[11] – DDS Request Types names

6.2.11.2 DDSXML_Request Class Enumerations

- enum DDSXML_Request::DDSXML_RequestTypes – Valid DDS Request Types
- enum DDSXML_Request::DDSXML_ImplementationRequestTypes – Valid DDS Implementation Request Types The internal progress objects must match up to these implementation types.
    - Catalog Request - URID
    - Standard Request - SDR/EDR/IP, RDR, ANC, AUX, DIARY
    - Catalog Query - Catalog Query
    - Temporal Request - SDR/EDR/IP, RDR, ANC, AUX, DIARY GEO Request - SDR/EDR/IP, RDR, DIARY
    - Temporal Geo Request - SDR/EDR/IP, RDR, DIARY

6.2.11.3 DDSXML_Request Class Functions

6.2.11.3.1 DDSXML_Request::getRequestType

DDSXML_RequestTypes DDSXML_Request::getRequestType (

)

This method returns the type of the Request.

Returns:

- DDSXML_RequestTypes the RequestType of the Request or 0 if none

6.2.11.3.2 DDSXML_Request::getRequestTypeName

std::string DDSXML_Request::getRequestTypeName (

)

This method returns the name of the Request Type.

Returns:

- std::string the RequestType of the Request

6.2.11.3.3 DDSXML_Request::getRequestName

std::string DDSXML_Request::getRequestName (

)

This method returns the name of the Request.

Returns:

- std::string the RequestType of the Request

6.2.11.3.4 DDSXML_Request::setRequestName

bool DDSXML_Request::setRequestName (

std::string requestName

)

This method sets the name of the Request.

Parameters:

- requestName The request name

Returns:

- bool True if OK, else false

6.2.11.3.5 DDSXML_Request::getRequestImplementationType

DDSXML_ImplementationRequestTypes

DDSXML_Request::getRequestImplementationType (

)

This method returns the implementation type of the Request.

Returns:

- DDSXML_ImplementationRequestTypes The Request Implementation Type of the Request or 0 if none

6.2.11.3.6 DDSXML_Request::getRequestImplementationName

std::string DDSXML_Request::getRequestImplementationName (

)

This method returns the implementation name of the Request.

Returns:

- std::string The Request Implementation Type

6.2.11.3.7 Deleted

6.2.11.3.8 DDSXML_Request::isTemplate

bool DDSXML_Request::isTemplate (

)

This method retrieves the template flag for this request. This will allow for any request in the future to be a template.

Returns:

- bool true = This request is a template false= This request is not a template

6.2.11.3.9 DDSXML_Request::addDestination

bool DDSXML_Request::addDestination (

std::string destinationID

)

This method adds a destination to the existing Request XML. The destination information is set according to the Destination object parameter. The NumberOf Destinations is also incremented by this method.

Parameters:

- destinationID The user defined destination ID to be added to the request.

Returns:

- bool - True if OK, else false if not

6.2.11.3.10 DDSXML_Request::getDestinations

std::vector<std::string> DDSXML_Request::getDestinations (

)

This method returns a vector of string objects that represents all destinations ID's currently entered into the request.

Returns:

- std::vector<std::string> The vector of strings that represent all of the destination IDs contained in the request.

6.2.11.3.11 DDSXML_Request::getNumberOfDestinations

int DDSXML_Request::getNumberOfDestinations (

)

This method returns the number of destinations that currently exist in the request.

Returns:

- int The number of destinations in the request.

6.2.11.3.12 DDSXML_Request::removeDestination

bool DDSXML_Request::removeDestination (

std::string destinationID

)

This method removes a user defined destination from the request.

Parameters:

- destinationID The user defined destination ID for the destination to be removed from the request or the destination ID was not found.

Returns:

- bool - True if an error occurred in executing the command

6.2.11.3.13 DDSXML_Request::destinationExists

bool DDSXML_Request::destinationExists (

std::string destinationID

)

This method check to see if a Destination ID already exists in the request.

Parameters:

- destinationID The string that contains the destination ID

Returns:

- bool true - if the Destination exists false - if the Destination does not exist

6.2.11.3.14 Deleted

6.2.11.3.15 DDSXML_Request::getImplementationTypes

static std::vector<DDSXML_ImplementationRequestTypes>
DDSXML_Request::getImplementationTypes (

DDSXML_RequestTypes requestType

)

This method returns a vector of implementation types that can be used for the Request type passed in. This is static so it can be called without needing an instance of this;

Parameters:

- requestType The request type

Returns:

- std::vector<DDSXML_ImplementationRequestTypes The vector of strings that represent all the possible DDSXML_ImplementationRequestTypes for a specific request type.

6.2.11.3.16 DDSXML_Request::getDestinationUsedFlag

bool DDSXML_Request::getDestinationUsedFlag (

)

This method returns the value of the used flag True is returned if the value is used in this request type.

Returns:

- bool True if used in this request type.

6.2.11.3.17 Deleted

6.2.11.3.18 DDSXML_Request::getTemplateUsedFlag

bool DDSXML_Request::getTemplateUsedFlag (

)

This method returns the value of the Template used flag True is returned if templates are used in this request type.

Returns:

- bool True if used in this request type.

6.2.11.3.19 DDSXML_Request::getSuspendTime

Int64 DDSXML_Request::getSuspendTime (

)

This method returns the value of the Suspend time. The time is in IET (microseconds).

Returns:

- Int64 The next Execution Time for the request.

6.2.11.3.20 DDSXML_Request::getNextExecutionTime

Int64 DDSXML_Request::getNextExecutionTime (

)

This method returns the value of the next execution. The meaning of this field is determined by the Request Server. The time is in IET (microseconds).

This is for the Request Servers internal use only and may not be set by the user.

Returns:

- Int64 The next Execution Time for the request.

6.2.12 DDSXML_StandardRequest Class Reference

This is the XML data class for the Standard Request. This class is responsible for storing and maintaining the state of a request in the system.

The Class diagram representing the DDSXML_StandardRequest Class is provided in Figure 6.2.12-1, DDSXML_StandardRequest UML Diagram.

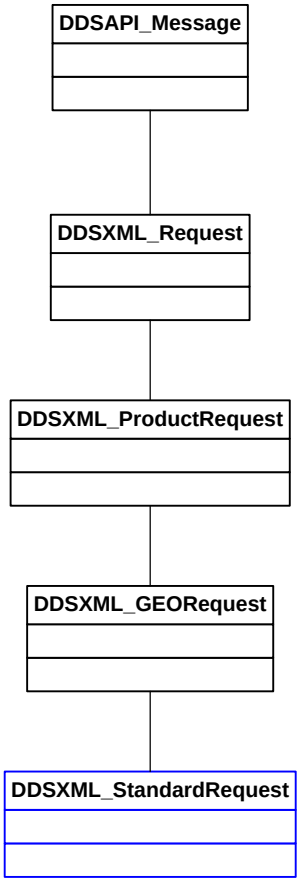


Figure 6.2.12-1, DDSXML_StandardRequest UML Diagram

6.2.13 DDSXML_SystemMessage Class Reference

This class is responsible for handling the system message XML. It creates, reads, writes and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML is validated then no invalid XML should be sent across the API.

The Class diagram representing the DDSXML_SystemMessage Class is provided in Figure 6.2.13-1, DDSXML_SystemMessage UML Diagram.

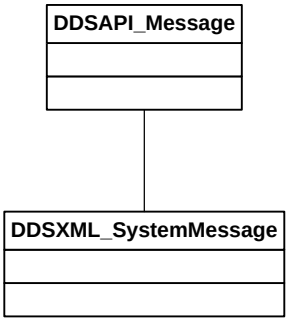


Figure 6.2.13-1, DDSXML_SystemMessage UML Diagram

6.2.13.1 DDSXML_SystemMessage Class Enumerations

- enum DDSXML_SystemMessage::DDSXML_SystemMessageSeverity – Valid System Message severity levels

6.2.13.2 DDSXML_SystemMessage Class Functions

6.2.13.2.1 DDSXML_SystemMessage::getMessage

std::string DDSXML_SystemMessage::getMessage (

)

This method retrieves the message.

Returns:

- std::string The message .

6.2.13.2.2 DDSXML_SystemMessage::setMessage

void DDSXML_SystemMessage::setMessage (

const std::string & message

)

This method sets the message.

Parameters:

- message The message .

6.2.13.2.3 SystemMessage::getTimeStamp

Int64 DDSXML_SystemMessage::getTimeStamp (

)

This method retrieves the TimeStamp. The time is in IET (microseconds).

Returns:

- Int64 The TimeStamp .

6.2.13.2.4 DDSXML_SystemMessage::setTimeStamp

void DDSXML_SystemMessage::setTimeStamp (

Int64 timeStamp

)

This method sets the TimeStamp. The time is in IET (microseconds).

Parameters:

- timeStamp The TimeStamp .

6.2.13.2.5 DDSXML_SystemMessage::setSeverity

void DDSXML_SystemMessage::setSeverity (

DDSXML_SystemMessageSeverity severity

)

This method sets the Severity.

Parameters:

- severity The Severity .

6.2.13.2.6 DDSXML_SystemMessage::getSeverity

DDSXML_SystemMessage::DDSXML_SystemMessageSeverity

DDSXML_SystemMessage::getSeverity (

)

This method retrieves the Severity from the XML.

Returns:

- DDSXML_SystemMessageSeverity The Severity .

6.2.13.2.7 DDSXML_SystemMessage::getStoreFlag

bool DDSXML_SystemMessage::getStoreFlag (

)

This method retrieves the Store Flag.

Returns:

- bool The Store Flag .

6.2.13.2.8 DDSXML_SystemMessage::setStoreFlag

void DDSXML_SystemMessage::setStoreFlag (

bool storeFlag

)

This method sets the Store Flag.

Parameters:

- storeFlag The Store Flag .

6.2.14 DDSXML_TemporalRequest Class Reference

This is the XML data class for the Temporal Request. This class is responsible for storing and maintaining the state of a request in the system.

The Class diagram representing the DDSXML_TemporalRequest Class is provided in Figure 6.2.14-1, DDSXML_TemporalRequest UML Diagram.

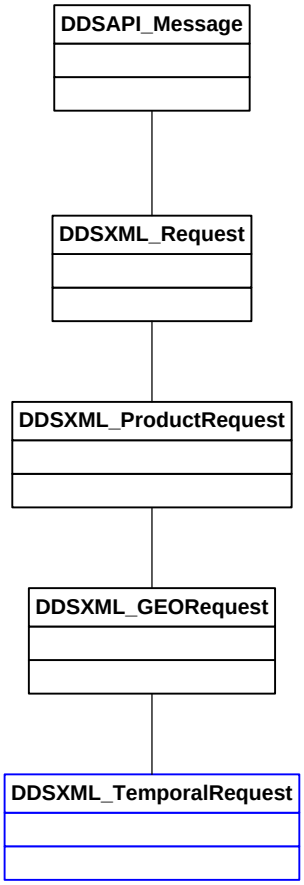


Figure 6.2.14-1, DDSXML_TemporalRequest UML Diagram

6.2.14.1 DDSXML_TemporalRequest Class Functions

6.2.14.1.1 DDSXML_TemporalRequest::getTemporalStart

Int64 DDSXML_TemporalRequest::getTemporalStart (

)

This method returns the temporal start for the request in Int64. The time is in microseconds since midnight. getTemporalStartUsedFlag() must return true for this field to be valid.

Returns:

- Int64 The request's temporal start

6.2.14.1.2 DDSXML_TemporalRequest::getTemporalDuration

Int64 DDSXML_TemporalRequest::getTemporalDuration (

)

This method returns the temporal duration for the request in Int64. The time is in microseconds. getTemporalDurationUsedFlag() must return true for this field to be valid.

Returns:

- Int64 The request's temporal duration

6.2.14.1.3 DDSXML_TemporalRequest::setTemporalStart

bool DDSXML_TemporalRequest::setTemporalStart (

Int64 temporalStart

)

This method sets the temporal start for the request in Int64. The time is in microseconds since midnight. This must be between getMinimumTemporalStart() and getMaximumTemporalStart(); getTemporalStartUsedFlag() must return true for this field to be valid.

Parameters:

- temporalStart The request's temporal start

Returns:

- bool True if set ok

6.2.14.1.4 DDSXML_TemporalRequest::setTemporalDuration

bool DDSXML_TemporalRequest::setTemporalDuration (

Int64 duration

)

This method sets the temporal duration for the request in seconds. This must be between getMinimumTemporalDuration() and getMaximumTemporalDuration(); getTemporalDurationUsedFlag() must return true for this field to be valid. The time is in

microseconds.

Parameters:

- duration The request's temporal duration

Returns:

- bool True if set ok

6.2.14.1.5 DDSXML_TemporalRequest::getMinimumTemporalStart

Int64 DDSXML_TemporalRequest::getMinimumTemporalStart (

)

This method returns the MinimumTemporalStart The time is in IET (microseconds).

Returns:

- Int64 The MinimumTemporalStart

6.2.14.1.6 DDSXML_TemporalRequest::getMaximumTemporalStart

Int64 DDSXML_TemporalRequest::getMaximumTemporalStart (

)

This method returns the MaximumTemporalStart The time is in IET (microseconds).

Returns:

- Int64 The MaximumTemporalStart

6.2.14.1.7 DDSXML_TemporalRequest::getMinimumTemporalDuration

Int64 DDSXML_TemporalRequest::getMinimumTemporalDuration (

)

This method returns the MinimumTemporalDuration The time is in IET (microseconds).

Returns:

- Int64 The MinimumTemporalDuration

6.2.14.1.8 DDSXML_TemporalRequest::getMaximumTemporalDuration

Int64 DDSXML_TemporalRequest::getMaximumTemporalDuration (

)

This method returns the MaximumTemporalDuration The time is in IET (microseconds).

Returns:

- Int64 The MaximumTemporalDuration

6.2.15 DDSXML_DataProductIDList Class Reference

This class is responsible for handling the data product List XML. It creates, reads, writes and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML is validated then no invalid XML should be sent across the API. This is used for the initial configuration from the Server. The request uses an ID list only.

The Class diagram representing the DDSXML_DataProductIDList Class is provided in Figure 6.2.15-1, DDSXML_DataProductIDList UML Diagram.

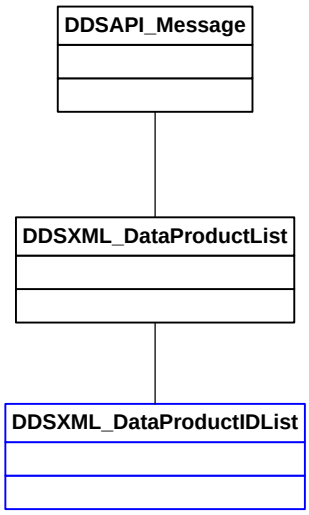


Figure 6.2.15 -1, DDSXML_DataProductIDList UML Diagram

6.2.15.1 DDSXML_DataProductIDList Class Attributes

- static std::string DDSXML_DataProductIDList::XML_DATA_PRODUCT_TAG – The XML tag to create this XML type.
- static std::string DDSXML_DataProductIDList::XML_DATA_PRODUCT_LIST_TAG – The

XML tag to create this XML type.

6.2.15.2 DDSXML_DataProductIDList Class Functions

6.2.15.2.1 DDSXML_DataProductIDList::operator=

DDSXML_DataProductIDList& DDSXML_DataProductIDList::operator= (

const DDSXML_DataProductIDList & dataProductIDList

)

Assignment operator=

Parameters:

- user The DDSXML_DataProductIDList that the data is to be copied from.

Returns:

- DDSXML_DataProductIDList A deep copy of the List

6.2.15.2.2 DDSXML_DataProductIDList::operator==

bool DDSXML_DataProductIDList::operator== (

const DDSXML_DataProductIDList & dataProductIDList

)

Compare operator==

Parameters:

- user The DDSXML_DataProductIDList that the data is to be compared to.

Returns:

- bool true if the two are equal.

6.2.15.2.3 DDSXML_DataProductIDList::validate

bool DDSXML_DataProductIDList::validate (

)

This method validates the data.

Returns:

- bool true if the data is valid.

6.2.15.2.4 DDSXML_DataProductIDList::validateFields

bool DDSXML_DataProductIDList::validateFields (

bool dpIDFlag

)

This method validates the XML.

Parameters:

- dpIDFlag If the flag is set validate this field

Returns:

- bool true if the data is valid.

6.2.15.2.5 DDSXML_DataProductIDList::addDataProduct

bool DDSXML_DataProductIDList::addDataProduct (

const DDSXML_DataProductID * dataProduct

)

This method adds the data Product to this list. Caller owns data sent in.

Parameters:

- dataProduct The data Product

Returns:

- bool true if the data is valid.

6.2.15.2.6 DDSXML_DataProductIDList::getDataProductsSet

virtual bool DDSXML_DataProductIDList::deleteDataProduct (

const std::string & dpID

)

Allows the caller to delete a data product from the list.

Parameters:

- dpID Deletes data product based on data product ID.

Returns:

- bool true if the data is valid.

6.2.15.2.7 DDSXML_DataProductIDList::getDataProductsSet

std::set<DDSXML_DataProductID* >
DDSXML_DataProductIDList::getDataProductsSet (

)

This method returns the DataProducts . Caller owns the data products returned.

Returns:

- set<DataProduct* > The Data Product's

6.2.15.2.8 DDSXML_DataProductIDList::getDataProducts

std::vector<DDSXML_DataProductID* >
DDSXML_DataProductIDList::getDataProducts (

)

This method returns the DataProducts . Caller owns the data products returned.

Returns:

- vector<DataProduct* > The Data Product's

6.2.15.2.9 DDSXML_DataProductIDList::getDataProductIDsSet

std::set<std::string > DDSXML_DataProductIDList::getDataProductIDsSet (

)

This method returns the DataProducts . Caller owns the data products returned.

Returns:

- set<std::string > The Data Product's

6.2.15.2.10 DDSXML_DataProductIDList::getDataProductIDs

std::vector<std::string> DDSXML_DataProductIDList::getDataProductIDs (

)

This method returns the DataProducts . Caller owns the data products returned.

Returns:

- vector<std::string > The Data Product's

6.2.15.2.11 DDSXML_DataProductIDList::getDataProduct

virtual DDSXML_DataProductID* DDSXML_DataProductIDList::getDataProduct (

const std::string & dataProductID

)

This method returns the dataProduct identified by the dataProduct ID if it exists in the system

Parameters:

- dataProductID The dataProductID to check for

Returns:

- int A pointer to the dataProduct . 0 - The dataProduct was not found. Valid pointer otherwise.

6.2.15.2.12 DDSXML_DataProductIDList::hasDataProductID

virtual bool DDSXML_DataProductIDList::hasDataProductID (

const std::string & dataProductID

)

This method returns true if the dataProduct ID exists in the system

Parameters:

- dataProductID The dataProductID to check for

Returns:

- int A pointer to the dataProduct . 0 - The dataProduct was not found. Valid pointer otherwise.

6.2.15.2.13 DDSXML_DataProductIDList::getNumberOfDataProducts

virtual int DDSXML_DataProductIDList::getNumberOfDataProducts (

)

This method returns the number of dataProducts for this user

Returns:

- int The number of dataProducts for this user

6.2.15.2.14 DDSXML_DataProductIDList::lockDataProductList

protected bool DDSXML_DataProductIDList::lockDataProductList (

)

This method locks the mutex controlling the DataProduct List object.

Returns:

- bool True if data filled in false if not.

6.2.15.2.15 DDSXML_DataProductIDList::unlockDataProductList

protected bool DDSXML_DataProductIDList::unlockDataProductList (

)

This method unlocks the mutex controlling the DataProduct List object.

Returns:

- bool True if data filled in false if not.

6.2.16 DDSXML_User Class Reference

This class is responsible for handling the user XML. It creates, reads, writes and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML is validated then no invalid XML should be sent across the API. The DDS system uses a username and user role as one unique user. All searches are base on the user name and user role as a key.

The Class diagram representing the DDSXML_User Class is provided in Figure 6.2.16-1, DDSXML_User UML Diagram.

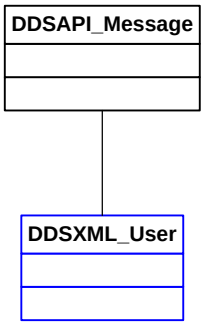


Figure 6.2.16 -1, DDSXML_User UML Diagram

6.2.16.1 DDSXML_User Class Functions

This class should not be used directly by the User API. Objects of this type cannot be accessed by the API.

6.2.17 DDSXML_UserList Class Reference

This class is responsible for handling the user list XML. It creates, reads, writes and extracts the data from/to the XML. It also allows for validation of the data. If the data and XML is validated then no invalid XML should be sent across the API.

The Class diagram representing the DDSXML_UserList Class is provided in Figure 6.2.17-1, DDSXML_UserList UML Diagram.

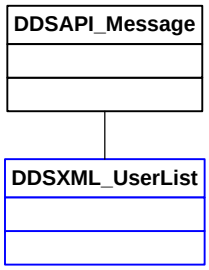


Figure 6.2.17 -1, DDSXML_UserList UML Diagram

6.2.17.1 DDSXML_UserList Class Attributes

- static std::string DDSXML_UserList::XML_USER_LIST_TAG – The XML header to create this message type.
- static std::string DDSXML_UserList::XML_USER_TAG – The XML tag to create this XML type.

6.2.17.2 DDSXML_UserList Class Functions

6.2.17.2.1 DDSXML_UserList::operator=

DDSXML_UserList& DDSXML_UserList::operator= (

const DDSXML_UserList & userList

)

Assignment operator=

Parameters:

- userList The DDSXML_UserList that the data is to be copied from.

Returns:

- DDSXML_UserList A deep copy of the User List

6.2.17.2.2 DDSXML_UserList::operator==

bool DDSXML_UserList::operator== (

const DDSXML_UserList & userList

)

Compare operator==

Parameters:

- userList The DDSXML_UserList that the data is to be compared to.

Returns:

- bool true if the two are equal.

6.2.17.2.3 DDSXML_UserList::validate

bool DDSXML_UserList::validate (

)

This method validates the data.

Returns:

- bool true if the data is valid.

6.2.17.2.4 DDSXML_UserList::validateFields

bool DDSXML_UserList::validateFields (

bool userListFlag

)

This method validates the XML.

Parameters:

- userListFlag If the flag is set validate this field

Returns:

- bool true if the data is valid.

6.2.17.2.5 DDSXML_UserList::getUsers

virtual std::vector< DDSXML_User*> DDSXML_UserList::getUsers (

)

Get the list of all users who the requestor has access to as a vector. Caller owns the list returned.

Returns:

- vector< DDSXML_User*> The vector that contains all users that the requestor has access to. The DDSXML_User objects referenced by this vector are owned by the caller and should be destroyed. This user may become invalid if a user is deleted from the list.

6.2.17.2.6 DDSXML_UserList::findUser

virtual bool DDSXML_UserList::findUser (

DDSXML_User * user

)

This method returns true if the user exists in the list.

Parameters:

- user The user to check for

Returns:

- bool True if the user is found in the list.

6.2.17.2.7 DDSXML_UserList::findUser

virtual DDSXML_User* DDSXML_UserList::findUser (

const std::string & userName,

const std::string & userRole

)

This method returns true if the user exists in the list. The caller owns the user returned.

Parameters:

- userName The name of the user to look for.
- userRole The Role of the user to look for

Returns:

- DDSXML_User* The user or 0 if not found in the list.

6.2.17.2.8 DDSXML_UserList::getNumberOfUsers

int DDSXML_UserList::getNumberOfUsers (

)

Get the list of all users as a vector. Caller owns the list returned.

Returns:

- int Number of users

6.2.17.2.9 DDSXML_UserList::lockUserList

protected bool DDSXML_UserList::lockUserList (

)

This method locks the mutex controlling the User List object.

Returns:

- bool True if data filled in false if not.

6.2.17.2.10 DDSXML_UserList::unlockUserList

protected bool DDSXML_UserList::unlockUserList (

)

This method unlocks the mutex controlling the User List object.

Returns:

- bool True if data filled in false if not.

6.2.18 Deleted

6.2.19 DDSXML_Longitude Class Reference

This is a storage container class for coordinate information.

The Class diagram representing the DDSXML_Longitude Class is provided in Figure 6.2.19-1, DDSXML_Longitude UML Diagram.

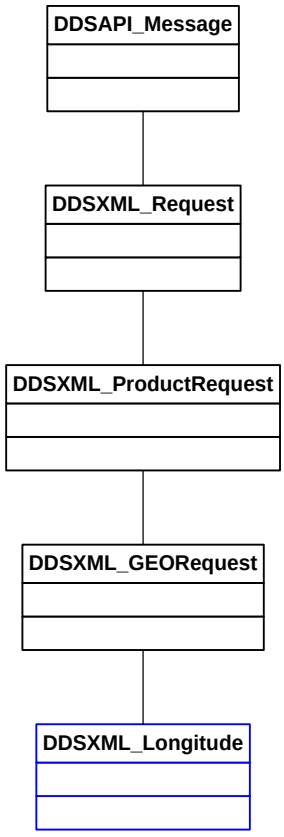


Figure 6.2.19 -1, DDSXML_Longitude UML Diagram

6.2.19.1 DDSXML_Longitude Class Functions

6.2.19.1.1 DDSXML_Longitude::setLowerRightLongitude

static DDSXML_Longitude* DDSXML_Longitude::setLowerRightLongitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Lower Right Longitude. The caller gets ownership of the Longitude

Parameters:

- isNegative Specifies if the LatLong is negative
- degrees The degrees value for the latitude or longitude as an integer.
- minutes The minutes value for the latitude or longitude as an integer.
- seconds The seconds value for the latitude or longitude as a float.

Returns:

- DDSXML_Longitude The Longitude or NULL if not valid.

6.2.19.1.2 DDSXML_Longitude::setUpperLeftLongitude

static DDSXML_Longitude* DDSXML_Longitude::setUpperLeftLongitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Upper Left Longitude The caller gets ownership of the Longitude

Parameters:

- isNegative Specifies if the LatLong is negative
- degrees The degrees value for the latitude or longitude as an integer.
- minutes The minutes value for the latitude or longitude as an integer.
- seconds The seconds value for the latitude or longitude as a float.

Returns:

- DDSXML_Longitude The Longitude or NULL if not valid.

6.2.20 DDSXML_Latitude Class Reference

This is a storage container class for coordinate information.

The Class diagram representing the DDSXML_ Latitude Class is provided in Figure 6.2.20-1, DDSXML_Latitude UML Diagram.

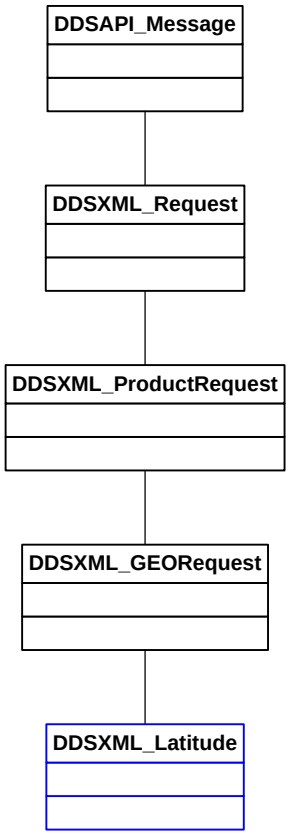


Figure 6.2.20 -1, DDSXML_Latitude UML Diagram

6.2.20.1 DDSXML_Latitude Class Functions

6.2.20.1.1 DDSXML_Latitude::setLowerRightLatitude

static DDSXML_Latitude* DDSXML_Latitude::setLowerRightLatitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Lower Right Latitude The caller gets ownership of the Latitude

Parameters:

- isNegative Specifies if the LatLong is negative
- degrees The degrees value for the latitude or longitude as an integer.
- minutes The minutes value for the latitude or longitude as an integer.
- seconds The seconds value for the latitude or longitude as a float.

Returns:

- DDSXML_Latitude The Latitude or NULL if not valid.

6.2.20.1.2 DDSXML_Latitude::setUpperLeftLatitude

static DDSXML_Latitude* DDSXML_Latitude::setUpperLeftLatitude (

bool isNegative,

int degrees,

int minutes,

float seconds

)

This method sets the Upper Left Latitude. The caller gets ownership of the Latitude

Parameters:

- isNegative Specifies if the LatLong is negative
- degrees The degrees value for the latitude or longitude as an integer.
- minutes The minutes value for the latitude or longitude as an integer.
- seconds The seconds value for the latitude or longitude as a float.

Returns:

- DDSXML_Latitude The Latitude or NULL if not valid.

APPENDIX A SYSTEM REQUIREMENTS

The Installation Guide for the NPOESS API is documented in the Installation Guide which accompanies the software as delivered by NPOESS. The Installation Guide file is identified in the distribution by the filename: INSTALL-JAVA, INSTALL-CPP, INSTALL-JMS. See the appropriate file details.

The NPOESS API is designed for use on the Microsoft® Windows® platform and on IBM AIX® (Advanced Interactive eXecutive) operating systems. The NPOESS API is developed and tested using the following Commercial Off-The-Shelf (COTS) Products

- Unix
    - IBM AIX® 5.3, 06-03-0732
    - Firefox 1.5.0.6
    - Citrix ICA Client 6.30.1095
    - StorNext Server/Client v.3.1.3
    - BEA Weblogic v. 8.1.3
- Windows
    - Windows XP Professional
    - Windows 2003 Service Pack 2
    - Microsoft IIS FTP Client
    - Internet Explorer
    - Citrix ICA Client Plugin
    - StorNext Server/Client (Server on Windows 2003 only) v. 3.1.3
    - BEA WebLogic Client

For the C++ APIs the following compilers are used:

- C++ APIs
  - Version 10 of IBM's VisualAge®
  - Version 7 of the Microsoft® Visual Studio® compiler
- Commercial Off-The-Shelf (COTS) Products
  - IBM AIX®
    - Version 2.6.0 of Apache[1] Xerces-C
    - Version 7.0 of Borland® VisiBroker®
    - Version 1.4 of Apache Axis
    - Version 0.9.8d of OpenSSL
    - Version 1.0.2 of log4cplus
    - Version 2.6.0 of Apache Xerces C++
    - Version 1.9.0 Xalan C++
  - Microsoft® Windows®
    - Version 2.6.0 of Apache Xerces
    - Version 7.0 of Borland® VisiBroker®Version 1.4 of Apache Axis
    - Version 0.9.8d of OpenSSL
    - Version 1.0.2 of log4cplus
    - Version 2.6.0 of Apache Xerces C++
    - Version 1.9.0 Xalan C++

[1] Apache refers to the Apache Software Foundation, which serves as a focal point for the development of standards-based XML solutions. Apache also provides feedback to international standards bodies such as Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C).

APPENDIX B DOCUMENT SPECIFIC ACRONYMS LIST

This table identifies and defines acronyms unique to this document. All other acronyms are listed and identified in the NPOESS Program Acronyms, D35838.

Table B-1, Document-Specific Acronym List

Acronym	Definition
AIX	Advanced IBM Unix
IETF	Internet Engineering Task Force
SEITO	System Engineering, Integration, Test and Operations
W3C	World Wide Web Consortium