

**GSFC JPSS CMO
February 18, 2015
Released**

Effective Date: January 26, 2015
Block/Revision 0200A

**Joint Polar Satellite System (JPSS) Ground Project
Code 474
474-00552-B0200**

Joint Polar Satellite System (JPSS) Java Application Programming Interface (API) User's Guide

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 2.0.0



**Goddard Space Flight Center
Greenbelt, Maryland**

National Aeronautics and
Space Administration

Check the JPSS MIS Server at https://jpssmis.gsfc.nasa.gov/frontmenu_dsp.cfm to verify that this is the correct version prior to use.

Joint Polar Satellite System (JPSS) Java Application Programming Interface (API) User's Guide

JPSS Electronic Signature Page

Prepared By:

Raytheon IIS SEIT for NASA JPSS Ground Project SEIT under NASA contract NNG10XA03C
(Electronic Approvals available online at https://jpssmis.gsfc.nasa.gov/mainmenu_dsp.cfm)

Approved By:

Rob Morgenstern
JPSS Ground Project Mission Systems Engineering Manager
(Electronic Approvals available online at https://jpssmis.gsfc.nasa.gov/mainmenu_dsp.cfm)

Daniel S. DeVito
JPSS Ground Project 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	Document Date	Description of Changes	Pages Affected
0200-	Dec 11, 2013	This version is the initial release per ECR-CGS-0247/474-CCR-13-1372 and was approved by the JPSS Ground ERB on the effective date shown.	All
0200A	Jan 26, 2015	This version incorporates ECR-CGS-0367/474-CCR-14-2097. This was approved by the JPSS Ground ERB on the effective date shown.	All

Table of TBDs/TBRs

Item No.	Location	Summary	Individual/ Organization	Due Date
47400552.TBD.0001	2.0	NASA to determine location where document can be obtained.	NASA	01/31/2013

Table of Contents

1. Introduction.....	1
1.1 Scope.....	1
1.2 Purpose.....	1
1.3 Organization	1
2. Related Documentation	2
2.1 Parent Documents	2
2.2 Applicable Documents	2
2.3 Information Documents.....	2
3. Overview	4
3.1 Granules.....	5
3.2 Hierarchical Data Format (HDF) Files	5
3.3 User Guide Delivery Format	5
3.4 Document Viewing Instructions	5
3.5 Call Sequence for a Standard Request.....	6
3.6 Call Sequence for a Retransmit	6
4. Client Operations	8
4.1 Initial Connection: How to Connect in order to Login.....	8
4.2 Logging into the System	8
4.3 Destination List Management	9
4.3.1 Specifying Destinations.....	9
4.4 Requesting Data Products from the Data Delivery Subsystem (DDS).....	10
4.4.1 Common User-Supplied Request Parameters.....	10
4.4.2 Request Examples	21
4.4.2.1 Specifying a Standard Request for SDR/EDR/IP/TDRs	21
4.4.2.2 Specifying a Standard Request for RDRs	21
4.4.2.3 Specifying a Standard Request for Spacecraft Diary, Telemetry, Dwell, and Dump RDRs.....	22
4.4.2.4 Specifying a Temporal Request for SDR/EDR/IP/TDRs	22
4.4.2.5 Specifying Standard Requests for Ancillary Data	23
4.4.2.6 Specifying Standard Requests for Auxiliary Data	24
4.4.2.7 Retransmit	25
4.5 Request Templates	27
4.6 Catalog Management	27
4.6.1 Data Catalog Queries.....	27
4.6.2 Data Catalog Requests.....	27
4.7 Supervisor Functions	29
5. Java API Documentation	30
5.1 Coding Conventions.....	30
5.1.1 Java Coding Conventions.....	30
5.2 Procedures for Client-side DDS API SSL Certificate Installation	31
5.3 Java API Module Documentation List.....	32

5.3.1 DDSAPI_Message Class Reference 32

6. API Module Code Examples 33

6.1 CreateStandardRequest.java 34

6.2 Create RetransmitRequest.java..... 37

6.3 Java SSL Properties and the PKI Properties File..... 40

List of Figures

Figure 5.2.1-1.DDSAPI_Message Class UML Diagram..... 35

List of Tables

Table 4.2-1 Login, Submit Request, and Logout Process..... 09

Table 4.3.1-1 Destination parameters Example 10

Table 4.4.1-1 Request Parameters 12

Table 4.4.2.1-1 Standard Request Parameters for SDR/EDR/IP/TDRs 21

Table 4.4.2.2-1 Standard Request Parameters for RDRs..... 22

Table 4.4.2.3-1 Standard Request Parameters for Spacecraft Diary RDRs 22

Table 4.4.2.4-1 Temporal Request Parameters 23

Table 4.4.2.5-1 Ancillary Data Standard Request Parameters..... 24

Table 4.4.2.6-1 Auxiliary Data Standard Request Parameters 25

Table 4.4.2.7-1 Retransmit Request Parameters 26

Table 4.4.2.7-2 Retransmit Request Parameters Example..... 26

Table 4.4.2.7-3 Retransmit Request Status 26

Table 4.6.2-1 Data Catalog Request Parameters 28

Table 4.6.2-2 Data Catalog Request Parameters Example 28

Table 5.1-1 JAVA Coding Conventions 30

1. Introduction

The Joint Polar Satellite System (JPSS) Java Application Programming Interface (API) User's Guide describes how users of JPSS environmental data can request and receive available data from the JPSS Interface Data Processing Segment (IDPS) through the IDPS Data Delivery Subsystem (DDS). This document is a description of a software to software interface. It includes information on setting a system up to use the interface functionality for request creation, management, and data delivery. The interface will enable users to manage delivery destinations and data requests on a user/user role basis. Example requests are provided to illustrate what is required by the system, and code examples illustrate how a user's software can define and submit such requests to the system.

1.1 Scope

This document is intended for any programmer who uses the DDS Java API. This document contains information about the DDS Java 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 Purpose

This guide is intended for any programmer who uses the DDS Java API.

1.3 Organization

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

Section 2 lists parent documents and related documents that were used as sources of information for this document or that provide additional background information to aid understanding of the interface implementations.

Section 3 is 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 describes the common and unique parameters required to generate requests for JPSS 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 provides the information for the Java version of the API.

Section 6 provides code examples for Standard and Retransmit Requests.

2. Related Documentation

The latest versions of all documents below should be used. The latest JPSS documents can be obtained from URL: (47400552.TBD.0001). JPSS documents have a document number starting with 470, 472 or 474.

2.1 Parent Documents

The DDS Java API User's Guide is written as a standalone document and as such does not have a Parent Document.

2.2 Applicable Documents

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

Document Number	Document Title
474-00410-B0200	Joint Polar Satellite System (JPSS) Common Ground System (CGS) to Comprehensive Large Array-Data Stewardship System (CLASS) Interface Control Document
474-00411-B0200	Joint Polar Satellite System (JPSS) Common Ground System (CGS) to National Environmental Satellite, Data, and Information Service (NESDIS) Environmental Satellite Processing Center (ESPC) Interface Control Document
474-00412-B0200	Joint Polar Satellite System (JPSS) Common Ground System (CGS) to National Aeronautics and Space Administration (NASA) Science Data Segment (SDS) Interface Control Document
474-00423-B0200	Joint Polar Satellite System (JPSS) Common Ground System (CGS) to Government Resource for Algorithm Verification, Independent Testing, and Evaluation (GRAVITE) Interface Control Document (ICD)
IC60917-SEIT-003	Common Ground System (CGS) to CGS Support Nodes (CSN) Interface Control Document (ICD)
474-00167	Joint Polar Satellite System (JPSS) Common Ground System (CGS) Requirements Document

2.3 Information Documents

The following documents are referenced herein amplify or clarify the information presented in this document. These documents are not binding on the content of this ICD.

Document Number	Document Title
474-00553-B0200	Joint Polar Satellite System (JPSS) Web Services User's Guide

Document Number	Document Title
UG60917-IDP-003	Joint Polar Satellite System (JPSS) Common Ground System (CGS) Interface Data Processing Segment (IDPS) Data Delivery Subsystem (DDS) Software (SW) Graphical User Interface (GUI) User's Manual Part 2, CDRL A032
474-00001-01-B0200	Joint Polar Satellite System (JPSS) Common Data Format Control Book – External Volume I Overview

3. Overview

This word document demonstrates the usage of the Java API through examples. The Java API provides a Java interface for users' Java software to use for communicating with JPSS IDPS/DDS. The technical content of the DDS Java API User's Guide is contained in a zip file with hash values available with the same name as this overview document (with .zip extension). This information is detailed in section 3.3 and 3.4 of this document. The purpose of this document is to provide guidance to users who are writing code or scripts that will request data from the JPSS IDPS/DDS. This document provides detailed descriptions for the following operations:

- logging onto 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 multifactor authentication as described in Section 4.2. The software submits the request supplying 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), certain Intermediate Products (IPs), Auxiliary Data, and Ancillary Data. A user may also request a granule from the data catalog. Only data specified in Mission Partner (MP) ICDs is made available for the user to request. Each request is associated with a user. The user's role may preclude them from receiving certain items, such as some IPs or Auxiliary Data.

Some request parameters include a timed process delay, request effectivity, aggregation, and delivery of repaired granules. Request effectivity bounds the data of interest relative to sensor observation time. The effectivity is 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 unless the request is a catalog query. 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.

Status of requests can be obtained by logging into the API and requesting status.

Some users, as determined by their role, 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 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 a 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 JPSS. 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, Volume I for more information.

3.2 Hierarchical Data Format (HDF) Files

The requested data and its associated metadata are wrapped and delivered in Hierarchical Data Format Group (HDF5) formatted files. If aggregation is not specified for JPSS 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.

3.3 User Guide Delivery Format

Due to the size and format of the DDS Java API Guides they will be delivered or downloaded via a ZIP file.

3.4 Document Viewing Instructions

To access the contents of the User Guide, the following steps must be completed:

1. Download **474-00552_JPSS Java API UG_0200A** to local drive.
2. Select "Extract All" or "Extract Here". Extract All will allow you to set a path for the extracted folder to go, Extract Here will uncompress the folder and leave it in the same location as the ZIP file.

The user may "click" on the index.html file to access the document main menu.

Once the files are extracted left click on the Hyper Text Markup Language (HTML) Document named "annotated" for the DDS Java API Class List. This is a list of the classes, structs, unions and interfaces for the API with brief descriptions, and the home point for navigation.

3.5 Call Sequence for a Standard Request

There are many different ways that application programmer may want to use DDS for requests. Outlined is one of them, which assumes the user knows the appropriate request parameters (e.g spacecraft, data product, and effectivity).

1. Call Login with mutual authentication.
 - a. Returns true if login was valid.
2. Call **addNewDestination** with the parameters specified in the DDS Java API guide to add a delivery destination for the data products. Alternatively, the user can call **getDestinations** to retrieve the list of already created destinations available to them, if any.
 - a. The destination Id is returned as a string if valid or an empty string if not valid.
 - b. This call is only needed for the first request and subsequent requests can use the same destination ID returned from this call or a destination ID from an existing destination by calling **getDestination**.
3. Call **createRequest** with the parameters specified in the DDS Java API guide.
 - a. This can be any valid request and implementation type combination.
 - b. createRequest returns an empty Request object that needs to be populated with required information prior to adding/submitting a request.
 - c. Returns the DDS request if valid else returns null.
4. Fill in the request with the proper fields, based on the request type.
 - a. The fields are described in the DDS Java API Guide.
 - b. Destination ID must match the one in Step 2, or must match any other destination ID that is valid for the user. A full list of valid destination IDs that may be used can be retrieved by calling getDestinations.
5. Call **addRequest** with the request from step 4.
 - a. Returns true if request was valid and all fields required were entered.
6. If any issues occur during processing, getSystemMessages may be called to acquire DDS error messages. These messages are useful for debugging purposes.
7. Call Logout.
 - a. Returns true.

3.6 Call Sequence for a Retransmit

Data can be requested for retransmission using an IDPS compliant file name. More information is provided for Retransmits in section 4.4.2.7, but the general call sequence is outlined here.

1. Call Login with mutual authentication.
 - a. Returns true if login was valid.
2. Call **addNewDestination** with the parameters specified in the DDS Java API guide to add a delivery destination for the data products. Alternatively, the user can call getDestinations to retrieve the list of already created destinations available to them, if any.

- a. The destination Id is returned as a string if valid or an empty string if not valid.
- b. This call is only needed for the first request and subsequent requests can use the same destination ID, or if an existing destination is desired, `getDestination` may be called to acquire the ID.
3. Create and fill in a List of **RetransmitRequestListElements**.
 - a. The fields are defined in the DDS Java API guide.
4. Call `processRetransmitList`.
 - a. This method takes the `destinationID` from step 2 and the List of `RetransmitRequestListElements` from Step 3.
 - b. This method returns a list of elements that represent the status of the initial retransmit list elements. It includes the filename originally submitted, a message in the case of a failed retransmit or a `requestID` if successful, and a submission status indicating true if the filename was processed for retransmit, false if it was not. A true submission status indicates that the request for the data was created, but does not indicate that the data has been delivered.
5. If any issues occur during processing, `getSystemMessages` may be called to acquire DDS error messages. These messages are useful for debugging purposes.
6. Call `Logout`.
 - a. Returns true.

Retransmit Reason Codes (from `DDSAPI_RetransmitReasonTypeEnum`)

<code>UNKNOWN_RETRANSMIT_REASON_TYPE</code>	A default value. Elements with this type are not processed by DDS.
<code>DELIVERY_FAILED_RETRANSMIT_REASON_TYPE</code>	Indicates failed delivery of the file.
<code>CRC_RETRANSMIT_REASON_TYPE</code>	Indicates Checksum failure.
<code>H_5_RETRANSMIT_REASON_TYPE</code>	Indicates H5 Corruption.
<code>METADATA_RETRANSMIT_REASON_TYPE</code>	Indicates incorrect metadata file contents. Elements with this type are not processed, as a software update is required to resolve this issue.

4. Client Operations

This section briefly describes the operations, classes and methods available to software developers. In summary, the client application must log into the DDS to create/manage destinations, create and submit a request, view the catalog, or manage or edit their requests. The client application may also inspect the system messages and product status while logged into the system.

How to reach the Java API/Web Services:

- NSOF A: <https://ddsa1.cgs/web/WSMessage?wsdl>
- NSOF B: <https://ddsb1.cgs/web/WSMessage?wsdl>
- NSOF I&T: <https://ddsint1.cgs/web/WSMessage?wsdl>
- CBU C: <https://ddsc1.cgs/web/WSMessage?wsdl>

4.1 Initial Connection: How to Connect in order to Login

Before a user can login to DDS Web Services they must have the following:

- A valid Non Person Entity (NPE) certificate.
- User Name, OSGroup, and IDPS Role entries in the InfSys_RolesDB for each environment the user will be logging in to. The NPE certificate will be linked to the User Name and used by the system to extract the IDPS Role information for the user. This linking will be accomplished during an enrollment process and must be completed prior to the user attempting to log in to the system. The enrollment process can be completed using the same NPE for more than one environment, allowing the user to connect to multiple strings as needed, however, only one string at a time.
- DDS_WS_URL – this is the URL of the DDS Web Service.
- PKI_PROPERTY_FILE information is detailed in Section 6.3

Refer to section 6 for examples of how to create an instance of the DDSAPI_Message.

Verify that:

- The client box has network connectivity to the Application Server
- The DDS Web Service is running on the Application Server
- The DDS Services are up and running
- The DDS Web Service is communicating with the DDS Services

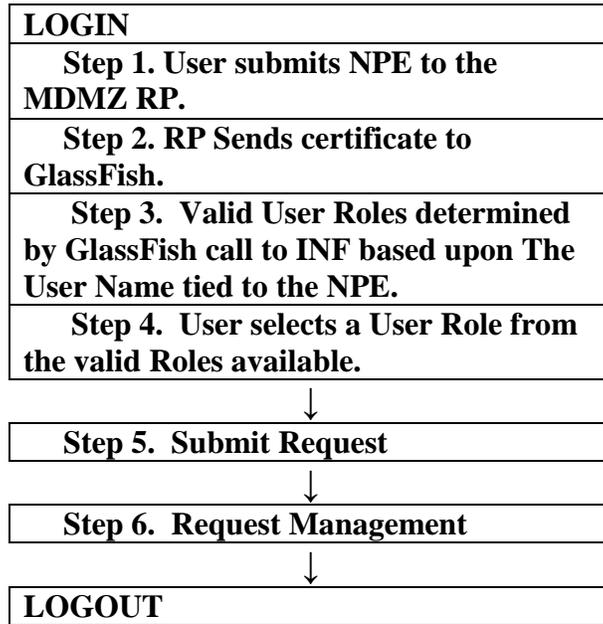
4.2 Logging into the System

In order to access any of the functions provided by the API, a login into the system must occur. This requires that mutual authentication be exchanged with the system. The DDS Java API user's NPE certificate is passed to the MDMZ reverse proxy (RP). The RP passes the certificate information to GlassFish, which then makes calls through INF to determine the user name and the roles that are available to that user name. The user then must choose their role, which determines what data may be accessed. Once the system authenticates and authorizes the login information, and a user role is selected, requests can be created, submitted, or managed. The user must remain logged in while creating, submitting, or managing requests. Requests are processed and the data products are shipped independent of whether the user or the user's software remains logged into the system.

Online Certificate Status Protocol (OCSP) Revocation will be checked for the NPE certificate within the Common Services. The OCSP repeater will relay the Revocation List.

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



Note: When a user exits the web service client's application without logout then tries to log in again, the system clears out the previous log in (i.e. it logs the user out and cleans up so there are no transient states retained from the old log in) and logs the user in again with the new information. No disconnect delay for the user.

4.3 Destination List Management

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

4.3.1 Specifying Destinations

Destinations are required for all request types except catalog queries, and define the location that the data requested is to be delivered to. At least one valid destination must be added to each request type that delivers data, up to a maximum of seven for each request. Any request that requires a destination but does not have at least one valid destination will be rejected. A user will be able to see destinations they have created, as well as destinations created by their Supervisor for their user role, when specifying destinations to add to a request.

Destinations are validated when they are added. A destination must include the host name, user name, password and path for non-local destinations, or the path for local

destinations. Users must use a Fully Qualified Domain Name with a known Landing Zone otherwise it will be rejected per Firewall Rules.

The user must have read, write, execute access to the destination path. Transfer type is defined by an enumeration for DestinationTransferTypeEnum. DDS User is the user that is returned from a getUser call.

The Destination shown in Table 4.3.1-1, Destination Parameters Example shows the parameters used to describe a Destination where data is to be delivered to the user.

Table 4.3.1-1, Destination Parameters Example

Destination	Destination	Destination Name	MyRequest
		Destination ID (Server Generated)	100000000001
		User Name	user
		User Password	****
		Path	C:/
		Transfer Type	FTPS
		User	DDS User

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

This section describes request functions for the JPSS IDPS/DDS software. Data request generation includes specifying the request type (see Table 4.4.1-1, Request Parameters) and supplying the request criteria, or parameters, required to generate the desired information. Requests consist of several parts, some common to all requests, and others 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 4.4.1-1, 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 JPSS 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 may modify the request parameters as desired, other than the system generated Request ID. 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.

4.4.1 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 or locations, compression, and effectivity of the request. Optional common code parameters include process delay and delivery of repaired data.

Table 4.4.1-1, 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 4.4.1-1, 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		Request 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 which give information about spacecraft health, attitude, and ephemeris.	
			RDR	These include science and 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 JPSS, and are required by JPSS algorithms (with the exception of sensor data) to achieve specific performance attributes per the CGS Requirements Document.	
			ANC	Ancillary Data. These data products are not produced by JPSS, but are required by JPSS algorithms to meet the attributes specified by the CGS Requirements Document for applicable JPSS Data Products.	

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
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
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	<p>The user specifies the destination of the requested data. Delivery of the data may require a username and password for some destinations.</p> <p>Prior to submitting the data request, the user must supply a destination. The destination can either be taken from an existing list of destinations submitted by either the user or the user's supervisor for the user's role, or can be a new destination created by the user for this request.</p> <p>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.</p>	Required
Compression	All	All	The user specifies whether to compress the HDF5 output	Required

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Effectivity	Standard, Temporal, Periodic, and Catalog Request	All	<p>Effectivity specifies the window of interest that the request is effective for. For Standard and Periodic Requests, this refers to the observation time (NPP/JPSS 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.</p> <p>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.</p> <p>The minimum start time is 949388400000000 (1988-02-01). The maximum start time (or end time) is 568028163299999 (2137-12-12 23:59:59).</p> <p>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:</p> <ul style="list-style-type: none"> • A standing request with start time after the stop time of the data. Standing requests have no end time specified. Any data that meets the criteria after the start time will be sent. • A one-time request with start time after the stop time of the data. 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. • A one-time request with end time before the start time of the data. 	Required

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
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 (12 hours).	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 un-aggregated (even if the initial delivery was aggregated). The repair delivery will include the repaired granule and its associated primary/secondary granules.	Optional

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Temporal	Temporal	SDR/EDR/IP, RDR, ANC, AUX, DIARY	<p>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).</p> <p>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.</p> <p>The delivered data might be slightly larger than the requested data because the delivery includes the entire granule that contains the specified time.</p>	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,	<p>The user may request NPP/JPSS 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.</p> <p>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.</p> <p>The delivered data might be slightly larger than the requested data because the delivery includes the entire granule that contains the specified coordinates.</p> <p>There exist 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.</p>	Optional

Common User-Supplied Parameters	Applicable Implementation Type	Applicable Request Type	Description	Required / Optional
Aggregation	Standard and Temporal	SDR/EDR/IP, RDR, DIARY	<p>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 JPSS 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).</p> <p>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.</p> <p>For a detailed description of the calculations involved, refer to the JPSS CDFCB-X Vol. I, 474-00001-01.</p>	Optional
Data Product ID	Standard, Temporal, Periodic, Catalog Request and Catalog Query	SDR/EDR/IP, RDR, DIARY, ANC, and AUX	<p>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.</p> <p>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.</p> <p>Each data product also has a sensor associated with it, but it is not formally part of the Data Product ID.</p>	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

4.4.2 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 JPSS Data Products, Ancillary Data, and Auxiliary Data.

4.4.2.1 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 JPSS Data Product types. Table 4.4.2.1-1, 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 hour; beginning 25 Jan 2003 07:00:00 – 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 4.4.2.1-1, 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	
Compression	true			

4.4.2.2 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 4.4.2.2-1 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 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 for data that is located in those GEO coordinates

that are supplied with the request.

Table 4.4.2.2-1 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	

4.4.2.3 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 4.4.2.3-1 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 4.4.2.3-1 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

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

Table 4.4.2.4-1, 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 6 minute duration. This is for data coming from the NPP platform. The geospatial setting is set.

Table 4.4.2.4-1, 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			

4.4.2.5 Specifying Standard Requests for Ancillary Data

Ancillary Data requests are requests for data not produced by JPSS, but required by JPSS algorithms to meet the attributes given in the Joint Polar Satellite System (JPSS) Common Ground System (CGS) Requirements Document (474-00167) (e.g., terrain height database or conventional surface and upper air observations).

The Ancillary Data request shown in Table 4.4.2.5-1, 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 4.4.2.5-1, 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

4.4.2.5.1 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.

4.4.2.6 Specifying Standard Requests for Auxiliary Data

Auxiliary Data requests are requests for data produced by JPSS, other than sensor data, which are required by JPSS algorithms to achieve the performance attributes given in Joint Polar Satellite System (JPSS) Common Ground System (CGS) Requirements Document (474-00167) (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 4.4.2.6-1, 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 4.4.2.6-1, Auxiliary Data Standard Request Parameters

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

4.4.2.6.1 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.

4.4.2.7 Retransmit

Retransmit functionality allows the user to recreate a previous request in cases where there was an error in retrieving the original data. The user is provided a convenience method called `processRetransmitList` for this purpose.

The `processRetransmitList` method accepts a single destination, where all the requested data will be sent, as well as a list of `RetransmitRequestListElement` objects.

Each `RetransmitRequestListElement` object is made up of a filename, a retransmit reason code, and optionally a URID. The filename represents the filename produced for the original request, the reason code indicates the reason the original request failed, and is of type

`RetransmitReasonTypeEnum`, and the optional URID, , which can be taken from a delivered DDR. Filename and URID for these elements may be found by examining the results of a DDR, CDDR or data shipment record. The filename is used to determine which product needs to be retransmitted for the request System analysis of the filename allows the system to regenerate a request based on the filename's content. Exact formats of the filenames varies depending upon the product type and is described in 474-00001-01_JPSS-CDFCB-X-Vol-I_0200-.docx

After composing the list, and either creating a new destination or selecting an existing destination, the user calls the `processRetransmitList` method, and will be returned a status in the form of a list of `RetransmitRequestStatus` objects. Each `RetransmitRequestStatus` object is made up of a filename, a requestID value, and a submission status indicator. The filename is the same as that in the originally submitted list (to allow correlation between the submitted list and the return list), the requestID (if available) indicates the request that was created by DDS to produce data, and the submission status indicates if the request was added to DDS for processing.

Note that returning from the call to `processRetransmitRequest` does not indicate that all the produced requests have completed processing. It only indicates that the `RetransmitRequestListElement` list has been processed and requests have been submitted to DDS for further handling.

Similarly, the submission status return value does not indicate delivery of desired files, only that a request was able to be created in order to retrieve and deliver the data. To determine the delivery status of a file submitted for retransmit, the user may choose to query DDS using the returned `requestID` value to call `findRequest` or other available methods to check the state/deliveries of the request.

Table 4.4.2.7-1, processRetransmitList Parameters

List of <code>RetransmitRequestListElements</code>	List of <code>RetransmitRequestElements</code>	Required
Destination ID	Destination ID of the desired delivery location for products resulting from the retransmit call. If more than one destination is desired, separate calls to <code>processRetransmitList</code> are required	Required

Table 4.4.2.7-2, RetransmitRequestListElement

Common User-Supplied Parameters	Description	Required / Optional
Filename	Filename resulting from the original request, which is desired for retransmit.	Required
Failure Reason	Reason that this file is being requested for retransmit. (failure reason for the original request)	Required
URID	URID of the data being requested for retransmit. This is used in some scenarios (based on product type and retransmit reason) to reduce processing time.	Optional

Table 4.4.2.7-3, RetransmitRequestStatus

Common User-Supplied Parameters	Description
Filename	Filename submitted for retransmit.
Request ID	Request ID (if available) of the request submitted within DDS to produce the desired products. May also contain text with failure information if the Submission Status is false.
Submission Status	True if a request was successfully submitted to DDS to deliver the desired products. False if an error condition occurred.

4.5 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.

4.6 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 to 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
Destination	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
Compression	All	The user specifies whether to compress the HDF5 output.	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
		Compression	true

Request Name
Request ID (Returned) 98765
URID 43132603-11104-9b9dea63-deb2216a
Destination h:/fileCatalog
Compression true

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. Java API Documentation

The DDS Java API is a set of libraries that can be used with any Java 1.7 JVM on either a Microsoft® Windows® or Linux 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 (in the dds.RequestAPI.Request package). All DDS Java API classes are in the RequestAPI.jar file, which is delivered as part of the DDS software delivery. There should only be one Message object instantiated for the user application.

5.1 Coding Conventions

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

5.1.1 Java Coding Conventions

All Java code follows the conventions in Table 5.1.1-1, Java Coding Conventions.

Table 5.1.1-1, Java Coding Conventions

Java 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.
Default constructors	All classes have a default constructor. Some of the default constructors may not be accessible to the user application, since those classes should not be instantiated in that manner.
Error exception reporting	All errors are reported as a Java exception derived from the Exception class. There are four types of exceptions that may be thrown: <ul style="list-style-type: none"> • RequestException • MessageException • ValidationException • DDSException DDSException is the base exception class for the other three, so catching the DDSException is sufficient when calling methods that throw any of these four exceptions.
toString()	The toString() is overridden in classes where necessary. This is done to facilitate the printing out of important contents of the particular object referenced.
equals() method	The equals() method exists in classes that can be compared. In particular, for data product and request related data.
Serializing Objects	Certain objects may be serialized. Those objects implement the java.io.Serializable interface. This interface is for data product and request related data.

Java Coding Conventions	
Application	Convention
Cloning Objects	Certain objects may be cloned. Those objects implement the java.lang.Cloneable interface. These are for data product and request related data.
Memory Management	All requests, templates, data products, and catalogs are managed through the Message object. These objects will be created and destroyed by the API. Do not attempt to manage these objects in your code. The Message object needs to be instantiated and deleted by the user application.

Refer to the JPSS Software Standards and Practices Manual (SSPM), MN60822-PMO-001, Appendix F, for additional details on coding guidelines.

5.2 Procedures for Client-side DDS API SSL Certificate Installation

Prerequisite:

- Obtain the signed certificate from the Certificate Authority (CA).

Overview:

- The examples below summarize the steps to add the signed certificates to the keystore.

Note:

Certificate name will contain the site, domain and renewal date so that multiple versions may be stored in case of a rollback.

Example: **myca.nesdis.ops.2010Feb14.crt**

Unix/Linux Java Client Import Procedures:

Copy or FTP the NPE certificate to a working directory of the target environment.

Example: **/tmp/working_dir/myca.nesdis.ops.2010Feb14.crt**

Import the certificate into the keystore.

```
> keytool -import -alias <key> -keystore cacerts -file <certificate>
```

Example:

```
> /usr/java6_64/bin/keytool -noprompt -import -alias mykey.`hostname`.`date`
+""%Y%m%d"" -keystore /usr/java6_64/jre/lib/security/cacerts -file
/tmp/working_dir/myca.nesdis.ops.2010Feb14.crt
```

Repeat the steps for all servers that will connect to IDPS via the DDS API.

Windows Java Client Import Procedures:

Copy or FTP the certificate to a working directory of the target environment.

Example: **E:\temp\working_dir\myca.nesdis.ops.2010Feb14.crt**

Import the certificate into the keystore.

```
E:\> keytool -import -alias mykey -file <certificate> -keystore \cacerts
```

Example:

```
E:\> E:\jpss\apps\jdk1.6.0_17\bin\keytool -import -alias mykey -file E:\  
temp\working_dir\myca.nesdis.ops.2010Feb14.crt -keystore  
E:\jpss\apps\jdk1.6.0_17\jre\lib\security\cacerts
```

Repeat the steps for all servers that will connect to IDPS via the DDS API.

5.3 Java API Module Documentation List

The Java 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 basic manipulations on catalog items and templates.

5.3.1 DDSAPI_Message Class Reference

This object is responsible for establishing and maintaining contact with the API Manager. This class is also responsible for handling commands that can be performed in the system. The 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. If there is a problem with the API use of a method, that method will exit with a false, a null string, or an empty vector. 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 objects will not have internal data members correctly populated so API will not recognize nor be able to use them.

6. API Module Code Examples

This section of the API contains sample code to `CreateStandardRequest.java` and `CreateRetransmitRequest.java`. The configuration files are provided in the `.jar` file. Note that in the examples below, the `PKI_PROPERTY_FILE` is the NPE certificate.

Compile the below examples from the directory they reside in, (in this example they reside in a directory called `API` using;

- The supplied `DDSJavaApi.jar` file supplied by DDS.
- The necessary COTS libraries.
 - `apache-log4j-1.2.16/log4j-1.2.16.jar`
 - `xerces-2_9_1/xercesImpl.jar`
 - `xalan-j_2_7_1/xalan.jar`
 - `Jakarta-commons-codec.jar`

The provided `DDSJavaApi.jar` file contains required configuration files needed to run the Client's java applications. Those configuration files are automatically extracted from the jar file to the user's working directory each time the client java application is executed.

The configuration files are:

1. `DDS_API_GuideList.cfg` contains a listing of all required xml files. Currently only contains `DDS_API_CFG.xml` as that is the only required xml file. Should not be modified by the JavaAPI user.
2. `DDS_API_CFG.xml` contains specific config parameters needed by the api. Currently contains two: `"DEBUG_DEST"` – defaulted to `"D_FILE"` – and `"DEBUG_PATH"` – defaulted to `"."`. Values given to the two config parameters can be changed by the user of the JavaAPI.
3. `CFGSchema.xsd` contains generic config parameters information, including names, types, and constraints. Used by `DDS_API_CFG.xml` to define what fields exist and are required in its configuration values. Should not be modified by the JavaAPI user.
4. `DDSJavaApi.java` contains information used by the API for converting time between IET and "normal" time. Should not be modified by the JavaAPI user.

```
>cd API
>mkdir classes
> javac -d ./classes -classpath DDSJavaApi.jar CreateStandardRequest.java
```

The compiled test can then be run using the necessary command line arguments and the compiled main class file.

```
>cd API
> java -classpath ./classes:DDSJavaApi.jar:log4j-1.2.16.jar:xercesImpl.jar:xalan.jar
CreateStandardRequest <pki_filename> <ws_URL>
```

6.1 CreateStandardRequest.java

```

import java.util.Vector;

import dds.RequestAPI.Exceptions.DDSAPI_MessageException;
import dds.RequestAPI.Message.DDSAPI_Message;
import dds.RequestXML.ExternalXMLData.DDSXML_DataProductID;
import dds.RequestXML.ExternalXMLData.DDSXML_Destination;
import dds.RequestXML.ExternalXMLData.DDSXML_DestinationTransferTypeEnum;
import dds.RequestXML.ExternalXMLData.DDSXML_ImplementationRequestTypeEnum;
import dds.RequestXML.ExternalXMLData.DDSXML_Request;
import dds.RequestXML.ExternalXMLData.DDSXML_RequestTypeEnum;
import dds.RequestXML.ExternalXMLData.DDSXML_StandardRequest;
import dds.RequestXML.ExternalXMLData.DDSXML_User;

/**
 * CreateStandardRequest This client code will connect to the DDS Web Service
 * and create a standard request.
 *
 */
public class CreateStandardRequest
{
    /**
     *
     */
    public static void main(String[] args)
    {
        if (args.length < 2)
        {
            System.out.println("Args: PKI_PROPERTY_FILE DDS_WS_URL");
            return;
        }
        DDSAPI_Message msg = null;
        try
        {
            msg = new DDSAPI_Message(args[0], args[1], false);

            String role = "IDP_OPERATIONS_CONTROL";
            DDSXML_RequestTypeEnum reqType =
            DDSXML_RequestTypeEnum.RDR_REQUEST_TYPE_ENUM;
            DDSXML_ImplementationRequestTypeEnum implType =
            DDSXML_ImplementationRequestTypeEnum.STANDARD_IMPL_REQUEST_TYPE_ENUM;

            if (!msg.login(role))
            {
                System.out.println("login failed");
                return;
            }

            DDSXML_User user = msg.getUser();

```

```

// Create the request
// This is an empty request of the proper request type and impl type
// It will have a valid request Id
// It is not inserted into DDS until a add is done
DDSXML_Request request = msg.createRequest(reqType, implType, user);

if (request == null)
{
    System.out.println("failed to create request");
        msg.logout(null);
        msg = null;
    return;
}

String id = request.getID();
System.out.println("created request: " + id);
DDSXML_StandardRequest stdReq = (DDSXML_StandardRequest) request;

//Get list of existing destinations
//and if there are any, take the first one
String destId = "";
Vector<DDSXML_Destination> dests = msg.getDestinations();
if (!dests.isEmpty())
{
    destId = dests.get(0).getID();
    System.out.println("Using existing destination "+destId);
}
else
{
    // Otherwise, create a destination
    // This creates a new local destination to the current directory
    // for delivery of files in a request
    destId = msg.addNewDestination(
        "foo_dest", //destination name
        null, //host name - not reqd for local
        "./", //path to write to on local system
        null, //ftpUserName - not reqd for local
        null, //ftpPassword - not reqd for local
        DDSXML_DestinationTransferTypeEnum.DESTINATION_TRANSFER_LOCAL_ENUM,
//transfer type
        user
    );

    if (!destId.isEmpty())
    {
        System.out.println("Destination "+destId+" created");
    }
}

if (destId.isEmpty())

```

```
{
    msg.logout(null);
    System.out.println("failed to find / create a destination");
    return;
}

        stdReq.addDestination(destId);

// Fill in the request with valid information
stdReq.setName("StandardTestRequest");

// start and end times in IET
stdReq.setStartTime(1730419233000000L); // 11/1/2012
stdReq.setEndTime(1730505633000000L); // 11/2/2012

    DDSXML_DataProductID dataProdId = new DDSXML_DataProductID("VIIRS-DIAGNOSTIC-
RDR", "NPP");

    stdReq.addDataProduct(dataProdId.getID());

    boolean added = msg.addRequest(stdReq, user);

    if (added)
    {
        System.out.println("Added request");
    }
    else
    {
        System.out.println("FAILED to add request");
    }

// Send a logout to the DDS Web Service
// This must logout since only one login is allowed at a time
// If a logout is not done then a login cannot be done until
// a disconnect timeout is done
    msg.logout(null);
    msg = null;
}
catch (DDSAPI_MessageException e)
{
    if (msg != null)
    {
        msg.logout(null);
        msg = null;
    }
    e.printStackTrace();
}
}
```

6.2 Create RetransmitRequest.java

```

import java.util.List;
import java.util.Vector;

import dds.RequestAPI.Exceptions.DDSAPI_MessageException;
import dds.RequestAPI.Message.DDSAPI_Message;
import dds.RequestXML.ExternalXMLData.DDSXML_Destination;
import dds.RequestXML.ExternalXMLData.DDSXML_DestinationTransferTypeEnum;
import dds.RequestXML.ExternalXMLData.DDSXML_RetransmitReasonTypesEnum;
import dds.RequestXML.ExternalXMLData.DDSXML_RetransmitRequestListElement;
import dds.RequestXML.ExternalXMLData.DDSXML_RetransmitRequestStatus;
import dds.RequestXML.ExternalXMLData.DDSXML_User;

/**
 * CreateRetransmitRequest This client code will connect to the DDS Web Service
 * and send a retransmit request.
 *
 */
public class CreateRetransmitRequest
{

    /**
     *
     */
    public static void main(String[] args)
    {
        if (args.length < 2)
        {
            System.out.println("Args: PKI_PROPERTY_FILE DDS_WS_URL");
            return;
        }

        DDSAPI_Message msg = null;
        try
        {
            msg = new DDSAPI_Message(args[0], args[1], false);

            // Login to the DDS Web Service
            String role = "ONSITE_TE";
            if (!msg.login(role))
            {
                System.out.println("login failed");
                msg.logout(null);
                msg = null;

                return;
            }

            DDSXML_User user = msg.getUser();

            //Get list of existing destinations

```

```

//and if there are any, take the first one
String destId = "";
Vector<DDSXML_Destination> dests = msg.getDestinations();
if (!dests.isEmpty())
{
    destId = dests.get(0).getID();
    System.out.println("Using existing destination "+destId);
}
else
{
    // Otherwise, create a destination
    // This creates a new local destination to the current directory
    // for delivery of files in a request
    destId = msg.addNewDestination(
        "foo_dest", //destination name
        null, //host name - not reqd for local
        "./", //path to write to on local system
        null, //ftpUserName - not reqd for local
        null, //ftpPassword - not reqd for local
        DDSXML_DestinationTransferTypeEnum.DESTINATION_TRANSFER_LOCAL_ENUM,
//transfer type
        user
    );

    if (!destId.isEmpty())
    {
        System.out.println("Destination "+destId+" created");
    }
}

if (destId.isEmpty())
{
    msg.logout(null);
    System.out.println("failed to find / create a destination");
    return;
}

Vector<DDSXML_RetransmitRequestListElement> retransList = new
Vector<DDSXML_RetransmitRequestListElement>();

// ANC with retransmit failure type, no urid
DDSXML_RetransmitRequestListElement elemANC = new
DDSXML_RetransmitRequestListElement();
elemANC.setFilename("off_NCEP-GFS-03HR-
ANC_GFS_NCEP_003f_20120514_201205141200Z_20120514154513Z_ee20120514180000Z_np.h5");

elemANC.setFailureReason(DDSXML_RetransmitReasonTypeEnum.H_5_RETRANSMIT_REASON_
TYPE_ENUM);
retransList.add(elemANC);

```

```

// ANC with URID and retransmit failure type
DDSXML_RetransmitRequestListElement elemANC2 = new
DDSXML_RetransmitRequestListElement();
elemANC2.setFilename("off_NCEP-GFS-03HR-
ANC_GFS_NCEP_003f_20120514_201205141200Z_20120514154513Z_ee20120514180000Z_np.h5");

elemANC2.setFailureReason(DDSXML_RetransmitReasonTypesEnum.H_5_RETRANSMIT_REASON
_TYPE_ENUM);
elemANC2.setUrid("52eadabd-3bf58-9d9b3c03-aab6ab88");
retransList.add(elemANC2);

// process the list
List<DDSXML_RetransmitRequestStatus> processRetransmitList = msg
    .processRetransmitList(retransList, destId);

boolean hasFailed = false;
if (!processRetransmitList.isEmpty())
{
    for (DDSXML_RetransmitRequestStatus statusEntry : processRetransmitList)
    {

        if (statusEntry.getSubmissionStatus())
        {
            System.out.println("Successfully submitted file "
                + statusEntry.getFilename()
                + " for resubmission with requestID "
                + statusEntry.getRequestID());
        }
        else
        {
            hasFailed = true;
            System.out.println("Failed to submit file "
                + statusEntry.getFilename()
                + " for resubmission with requestID "
                + statusEntry.getRequestID());
        }
    }
}

msg.logout(null);
msg = null;
}
catch (DDSAPI_MessageException e)
{
    e.printStackTrace();
}

if (msg != null)
{
    msg.logout(null);
}

```

```

    msg = null;
  }
}
}

```

6.3 Java SSL Properties and the PKI Properties File

Purpose: The interface with DDS requires two-way SSL authentication. The client software needs to point to an appropriately signed client certificate as well as an appropriately configured truststore. For Java applications, these files are in Java Keystore (jks) format and are passed in as Java properties.

<u>Property</u>	<u>Content</u>
javax.net.ssl.keyStore	path to client keystore (jks format)
javax.net.ssl.keyStorePassword	password of client keystore
javax.net.ssl.trustStore	path to truststore (jks format)
javax.net.ssl.trustStorePassword	password of truststore

Deployment.security.validation.crl.url URL to certificate revocation list (optional - only necessary for CRL checks)

The Java properties can be passed in through various mechanisms depending on user preferences. Possible options include `System.getProperties().setProperty()`, `System.getProperties().load()` and passing in as jvm arguments (i.e. `-Djavax.net.ssl.keyStorePassword=myPassword`).

The sample code above loads in the SSL properties through an external properties file. This is suggested since it prevents passwords from being hardcoded or passed in as arguments which can be seen via the "ps" command.

Sample **pki.properties** file content

```

javax.net.ssl.keyStore      /etc/pki/tls/certs/MP_ClientApp.jks
javax.net.ssl.keyStorePassword  keystorePW
javax.net.ssl.trustStore    /etc/pki/java/cacerts.jks
javax.net.ssl.trustStorePassword  tsPassword
deployment.security.validation.crl.url https://pki.treas.gov/NASA_Operational_CA1.crl

```

Since the passwords are plaintext, the file permissions on `pki.properties` should be set to prevent unauthorized access. Suggested Linux permissions are 400.