GSFC ESDIS CMO
October 1, 2021
Released

423-ICD-027, Original
Earth Science Data and Information Systems (ESDIS) Project, Code 423

Science Data Transfer Protocol (SDTP) Interface Control Document (ICD)

Goddard Space Flight Center
Greenbelt, Maryland

National Aeronautics and
Space Administration

Science Data Transfer Protocol (SDTP)
Interface Control Document

Signature/Approval Page

Prepared by:

Signature obtained on file | ***10/01/2021***
Robert Wolfe | Date
Terrestrial Information Systems Laboratory Chief
NASA GSFC Code 619

Reviewed by:

Signature obtained on file | ***10/01/2021***
Evelyn Ho | Date
SDS Manager
NASA GSFC Code 423

Reviewed by:

Signature obtained on file | ***10/01/2021***
Karen Michael | Date
EOSDIS System Manager
NASA GSFC Code 423

Approved by:

Signature obtained on file | ***10/01/2021***
Andrew Mitchell | Date
ESDIS Project Manager
NASA GSFC Code 423

Signatures available in B32 Room E148
online at: / https://ops1-cm.ems.eosdis.nasa.gov/cm2/

Preface

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

Any questions should be addressed to: esdis-esmo-cmo@lists.nasa.gov

ESDIS Configuration Management Office (CMO)
NASA/GSFC
Code 423
Greenbelt, Md. 20771

Abstract

This document identifies the Science Data Transfer Protocol (SDTP) as an Earth Science Data and Information System (ESDIS) standard interface mechanism. The SDTP is used for the transfer of data from one ESDIS element to another. For example, the SDTP mechanism would be used for an on-going transfer of science data from a Science Investigator-led Processing System (SIPS) to a Distributed Active Archive Center (DAAC). The subscriber pulls data from the provider using Hypertext Transfer Protocol Secure (HTTP) over Transport Layer Security (TLS) (HTTPS) methods.

Keywords: ESDIS, SDTP, SIPS, DAAC, HTTPS

Change History Log

Revision	Effective Date	Description of Changes (Reference the CCR & CCB Approval Date)
Original, Rev-	10/01/2021	CCR 423-ESDIS-321; CCB Approved 08/19/2021 Pages: Baseline Cover through 20; see DCN for details

Table of Contents

1 INTRODUCTION 1
1.1 Purpose 1
1.2 Scope 1
1.3 Related Documentation 1
2 OVERVIEW 3
2.1 Key Characteristics 3
2.2 Prerequisites 3
2.3 Data Flow Overview 4
3 DATA FLOW DETAILS 6
3.1 Authentication 6
3.2 File List 6
3.3 Common Tags 9
3.4 File Transfer 10
3.5 File Transfer Acknowledgment 10
3.6 Extra Fields 11
3.7 Related File Groups 11
3.8 Polling 12
3.9 File List Paging 13
3.10 File Transfer Prioritization 14
3.11 Register Client Certificate 14
4 ERROR HANDLING 15
4.1 Incorrect File List 15
4.2 File Transfer Error 15
4.3 Related File Errors 16
4.4 Incorrect Request (Code 400) 16
4.5 File Authentication Error (Code 401) 16
4.6 Resource Error (Codes 403 and 404) 16
4.7 Too Many Requests (Code 429) 17
5 INTERFACE PARAMETERS 18
6 RUNNING THE PDR PROTOCOL ON TOP OF SDTP 19
Appendix A Acronyms 20

List of Figures

Figure 2-1. Data Flow. 4
Figure 3-1. Example file list. 7
Figure 3-2. Example group field. 12

List of Tables

Table 3-1. File list fields. 7

Table 3-2. Commonly used tags. 9
Table 3-3. Group fields. 17
Table 3-4. Pagination query parameters. 14
Table 4-1. HTTPS error status responses. 16
Table 5-1. Interface control parameters and their default values.. 19

Check ESDIS CM library, https://ops1-cm.ems.eosdis.nasa.gov/cm2/, to verify that this is the correct version prior to use.

1 INTRODUCTION

1.1 Purpose

The Polling with Delivery Record (PDR) protocol has served Earth System Data and Information System (ESDIS) well for decades, but as File Transfer Protocol (FTP) and Secure FTP (SFTP) is being phased out, a new method for transferring files between ESDIS elements is needed. The purpose of the Science Data Transfer Protocol (SDTP) is to provide an up-to-date mechanism for this data transfer using Hypertext Transfer Protocol (HTTP) over Transport Layer Security (TLS) (HTTPS) methods between a data provider and a data subscriber. For instance, this protocol would be used to transfer data from a Science Investigator-led Processing System (SIPS) to a Distributed Active Archive Center (DAAC).

1.2 Scope

This document was developed to be used as a standard protocol for exchange of data between ESDIS elements. These elements may be in the NASA Earth Science Cloud, at a NASA data center (including an on-premises cloud), or at another science data provider or subscriber.

The primary SDTP users are ESDIS data providers and subscribers that have sustained data flows and currently use the PDR protocol, had planned to use the PDR protocol in the future, or expect to establish new sustained data flows. ESDIS data providers are typically science data producers, e.g. SIPS, Science Data System (SDS), Earth Observing System (EOS) Data and Operations System (EDOS). ESDIS data subscribers are primarily DAACs. Note that sometimes the roles are reversed with a DAAC providing input data to a science data producer. Also, some science data producers may subscribe to data produced by other producers (e.g. VIIRS L1 (Level 1) and Land SIPS flow to the VIIRS Atmos. SIPS).

This protocol could also be used for sustained data flows between DAACs and other (outside) data providers or subscribers.

There are two related ESDIS file transfer mechanisms. The PDR protocol is the one that is most closely related and because of its ubiquity, it will likely co-exist with this new protocol for some time. The recently developed Cloud Notification Mechanism (CNM) is focused on file transfers with a Cloud and so complements this activity, but it does not meet the immediate need to replace the PDR protocol for the large number of systems not in the ESDIS Cloud.

Some existing PDR Protocol users may want to replace SFTP with SDTP but also continue using the PDR Protocol. Because using just SDTP is simpler and provides the same basic functionality as the PDR Protocol, this is not the preferred approach. However, an interim approach may be needed to enable legacy systems to move away from FTP/SFTP quickly and minimize disruptions of existing systems. Section 6 discusses an approach to do this where the data provider and subscriber each run both a SDTP server and client.

1.3 Related Documentation

The latest versions of all documents below should be used. The latest ESDIS Project documents can be obtained from Universal Resource Locator (URL): https://ops1-cm.ems.eosdis.nasa.gov. ESDIS documents have a document number starting with either 423 or 505. Other documents are

available for reference in the ESDIS project library website at https://doclib.eosdis.nasa.gov/ unless indicated otherwise.

Number	Title
423-41-57	ICD between the ECS and the SIPS, Volume 0, Interface
423-ICD-015	Cloud Notification Mechanism Interface Control Document

2 OVERVIEW

2.1 Key Characteristics

The defining characteristics of the SDTP are as follows:

- File lists and files are pulled by the subscriber.
- SDTP is designed to work for any file format/type, e.g. science data, science metadata, browse imagery.
- A file list may contain many files to minimize the overhead of obtaining new file lists.
- Subscriber acknowledges each successful file transfer.
- File list is a JavaScript Object Notation (JSON) object.
- Required fields in the file list are minimized.
- A set of commonly used *tags* is defined.
- Additional *tags* can be added as needed for each provider/subscriber pair.
- X.509 certificates are used for authentication.
- Focused on transferring files between the provider and subscriber.
- Not focused on transferring metadata or other information about the file.
- Standard way to group related files.
- HTTPS responses are used to indicate status (no other status message responses are needed).
- Handling of non-common errors is done *out-of-band*, via email or other means (see details in Section 4).
- File size is only limited by the HTTPS protocol.

2.2 Prerequisites

Both the provider and subscriber of the SDTP agree on the following:

- Provider URL for the HTTPS file transfer.
- Certificate Authority for authentication.
- Subscriber certificate Distinguished Name.
- A set of *tags* and corresponding valid values.
- A set of *extra* fields and corresponding valid values.
- Values for a set of parameters that control the interface, such as, the maximum number of files in a list (see Section 5).
- Points of contact.

This information is documented in the Operations Agreement (OA) between the provider and subscriber.

Tags and their valid values are used to control the data that is transferred from a provider to a specific subscriber (see the Common Tags in Section 3.3). For example, the OA may limit the value of the Earth Science Data Type (ESDT) *tag* to just the products produced by a SIPS (provider) that are meant to flow to a specific DAAC (subscriber).

2.3 Data Flow Overview

The interface only supports a pull mode of operation where the subscriber pulls the file lists and file contents from the provider.

The first step is for the subscriber and provider to use a certificate to establish a secure HTTPS connection. Then, to begin the file transfer, the subscriber requests a list of files from the provider using a set of *tags*. The subscriber then pulls all of the files in the list and acknowledges receipt of each file.

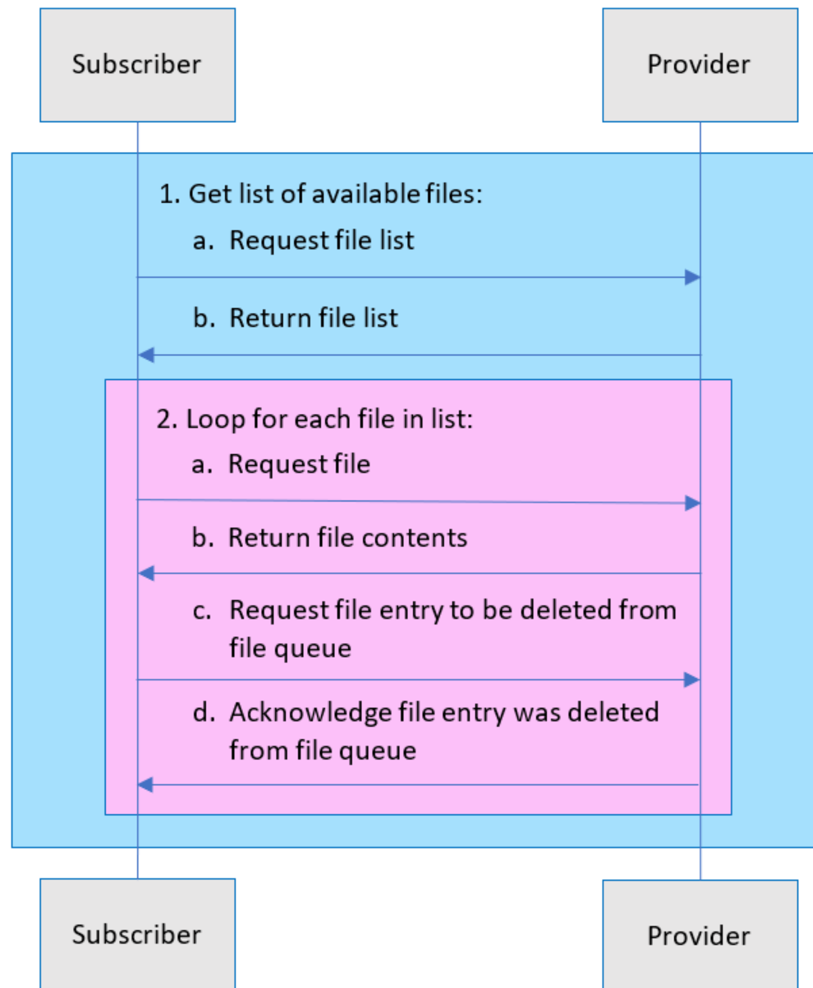

Figure 2-1. Data flow.

The provider stages a file to be transferred to a subscriber and assigns a unique *fileid* to the file (not shown). The provider adds the file to a queue of files to be transferred. Each file entry in the queue has a set of *tags* and values.

The individual steps that are involved in the data transfer (Fig. 2-1) are:

1. To obtain a list of available files:
    a. The subscriber performs a HTTPS GET with a set of *tags* and values.

   b. The provider returns a JSON object containing a list of files ready to be transferred. Only files for the subscriber that have a matching set of *tags* and values are included in the list.
2. For each file in the list:
   a. The subscriber requests the file contents by performing a HTTPS *GET* containing the *fileid*.
   b. The provider returns the file contents.
   c. The subscriber acknowledges the file transfer by performing a HTTPS *DELETE* containing the *fileid* of the file.
   d. The provider removes the file from the queue for the subscriber and returns a HTTPS *Success* status.

Most file transfer errors are handled using standard HTTPS error codes. Other errors, such as problems with the file list contents, are handled by email communication between the subscriber and provider. Section 4 discusses in detail the types of errors and approaches for handling them.

3 DATA FLOW DETAILS

The following discusses the details of the data transfer.

3.1 Authentication

Authentication is performed by using a trusted Certificate Authority (CA). Two X.509 public key certificates are required:

1. A normal HTTPS server certificate that guarantees the subscriber is talking to the actual provider URL.
2. A certificate for each subscriber that connects to a provider.

These HTTPS certificates are used to guarantee the identities of the provider and subscriber and encrypt the communication between them. The OA between the provider and subscriber will spell out the trusted CA that guarantees identities. ESDIS trusted partners, aka. providers and subscribers, will use client certificates obtained from a CA recommended by ESDIS. The provider server certificates are obtained either through the normal ESDIS NASA Access Management System (NAMS) server certificate workflow or from a well known CA. The subscribers will obtain the client certificate directly from the ESDIS recommended CA. Once the subscriber certificate is obtained, the subscriber will provide the certificate's Distinguished Name (DN) to the provider. A subscriber may use the same certificate for multiple providers. As described in Section 3.11, the subscriber may register a client certificate with the provider using an HTTPS PUT request.

The DN from the subscriber certificate is used by the provider as a unique identifier for the subscriber. This enables the provider to limit the subscriber's access to data sets that have been agreed upon by the provider and subscriber.

3.2 File List

The file list is requested by the subscriber by polling the provider URL (see Section 3.8 for the polling details).

The file list GET request typically contains a set of *tags*/values that limits the file list to only those with the appropriate *tags*. Section 3.3 describes a set of commonly used *tags*.

An example of a file list GET query is:

```
curl  -X GET "https://provider.org/sdtp/v1/files?stream=prod&ShortName=INS02A"
      -H "Accept: application/json"
```

This is an example of a resulting file list:

```
{
  "files": [
    {
      "fileid": 1342,
```

```
      "name": "file1.txt",
      "checksum":
  "sha256:f4c96f1f144f083485e8a4ea490cb605a7d0f2ffb7a11ec48e97a9e1631aa079",
      "size": 5678,
      "expires": "2020-03-30",
      "tags": {
        "stream": "prod",
        "ShortName": "INS02A",
        "Version": "061"
      }
    },
    {
      "fileid": 1355,
      "name": "file2.txt",
      "checksum":
  "sha256:ca7316a6bdba23870508ae72c53872bfc0a87520cbe3a88679130a81f400d5ae",
      "size": 15,
      "expires": "2020-03-30",
      "tags": {
        "stream": "prod",
        "ShortName": "INS02A",
        "Version": "061"
      }
    }
  ]
}
```


Figure 3-1. Example file list.

The file list JSON object returned contains fields for each file (Table 3-1). The *tags* and *extra* fields are optional, whereas the other fields are required to be present (and not null).

Table 3-1. File list fields.

Name	Type	Description	Notes
fileid	positive integer	Unique file identifier.	This is unique value assigned to the file by the provider and shall not be reused, e.g. *1342*
name	string	File name. Only the name – no directory information.	Maximum length is 256 characters, e.g. *file1.txt*
checksum	string	File checksum value with a checksum type prefix.	Maximum length is 256 characters, e.g. *sha256:f4c96f1f144f083485e8a4ea490cb605a7d0f2ffb7a11ec48e97a9e1631aa079*
size	positive integer	File size in bytes.	e.g. *5678*
expires	string	File expiration date in ISO-8601 format. Expires at the end of day (midnight GMT).	e.g. *2020-03-30*

tags	tag list	List of tags/values for this file.	Optional field.
Extra	extra values list	List of additional information for this file.	Optional field.

A successful HTTPS response has a status *Code 200 - Success* and returns a JSON object containing a list of files ready for retrieval. The HTTPS response header also includes a SDTP Transaction Identifier (*SDTP-TransactionID*), a Universally Unique Identifier (UUID), that is unique to each GET request. The *SDTP-TransactionID* is meant to be used for debugging. It uniquely identifies each event (for example in the HTTPS log files). Responses for various error conditions are discussed in Section 4.

If no files are available when the request is made, the response will be *Success* with an empty file list.

The maximum number of the files in the list is limited by an agreement between the provider/subscriber pair (see Section 5 on Interface Parameters) and/or by pagination (see Section 3.9 on File List Paging). The provider orders the entries in the subscriber queue by when they are entered into the queue and always provides them to the subscriber in that order (i.e. first-in first-out). *Tags* and *extra* fields can be used by the subscriber to help prioritize the order that files are transferred (see Section 3.10 on File Transfer Prioritization).

The *fileid* is a positive integer (limited to 15 digits) and is unique within the provider's SDTP server. The *fileid* value is assigned sequentially as each file entry is added to the provider's server queue.

The checksum type is something that is negotiated between the provider and subscriber (Section 5). In the near future, many systems will use *sha256*.

When using *curl*, the status information can be obtained using the options: *--write-out %{https_code}* to return the HTTPS code and *--verbose* option to return the headers. If either of these options are used, the *--output <file_name>* option to write the (non-header) response to an output file. More information on *curl* can be found at https://curl.haxx.se/docs/manpage.html.

The *expires* field is used by the provider to indicate to the subscriber the length of time that a file entry will remain available in the subscriber's queue. One use of this field is for providers with limited disk space used for staging files for subscribers. In this case, it is useful to let the subscriber know that the staged files are only available for a limited time period. The *expires* value could be a fairly large amount of time, e.g. several months, from when the file entry is added to the subscriber queue. Note that when the subscriber acknowledges receipt of the file, the provider will typically delete the file entry from the subscriber queue and if a staging disk is used, may remove the file from the staging disk. The default value for this field is a tunable parameter and agreed to by the subscriber/provider pair (see Section 5 Interface Parameters).

The provider should monitor each subscriber queue to determine what file entries are near their expiration date and have not yet been transferred, that is, the subscriber has not yet provided a file transfer acknowledgement. In addition, a good practice is for the provider to check each subscriber queue daily and determine if any file transfers are taking longer than normal. For instance, if most files transfers occur in a few hours and a file transfer has not been acknowledged for several days, then the file transfer could be considered late. For any file entries in this state, the provider should work with the subscriber to resolve any issues that may have caused the transfer to be delayed.

The *tags* field contains a list of *tag* values associated with each file entry by the provider. See Section 3.3 for a list of commonly used *tags*.

The *extra* field is a JSON object that contains additional information associated with that file entry by the provider (see Section 3.6 on Extra Fields). This field is not used to filter file entries like the *tags* field. The contents of this field are typically negotiated between the provider and subscriber. One common use of this field is to group related files (see Section 3.7).

Some subscribers may prefer the file list in a different format such as Comma-Separated Values (CSV). In this case the subscriber can use a tool such as *jq* (https://stedolan.github.io/jq) to convert the JSON response to CSV format.

3.3 Common Tags

The *tags* must be negotiated between the provider and subscriber, including the list of valid values for each *tag*. The tag/value pairs are used to filter which files are delivered to a subscriber, and can also be used to filter the file list, for example to expedite delivery of some types of data over others. A set of commonly used *tags* is given in Table 3-2.

Tags and their valid values may also be *stream* specific. For example, a test *stream* may have a different set of *tags* and/or valid values than a production *stream*.

Table 3-2. Commonly used tags.

Name	Description	Example
stream	Data stream identifier.	prod
ShortName	Collection short name, aka. Earth Science Data Type (ESDT).	INS03A
Version	Collection version identifier.	061

Tags and their values are strings and case sensitive. When using *tags* that are metadata fields in the ESDIS Universal Metadata Model (UMM), a best practice is to use the UMM metadata field name (with the same capitalization), e.g. *ShortName*. Because *tags* and values are passed as part of the *GET* query, avoid spaces and other special characters (ones that need additional URL encoding). A best practice is to use dashes (-) as separators within a *tag* value (e.g. *test-1*).

One key tag is the *stream* tag which is used to help separate individual *streams* of data being transferred. For instance, there may be one *stream* for forward processing, a second one for reprocessing and a third for testing. For this case, the *stream* tag values could be *prod* for forward processing, *reproc* for reprocessing and *test* for test. A numerical suffix (i.e. *-n*) could be included if there are multiple streams of one type (e.g. *test-1*, *test-2*).

Another possible use of a *tag* would be to include a *date* tag. This would allow the subscriber to prioritize pulling files with a specific date value(s). For instance, when there is a backlog, the subscriber may want to transfer data that had been most recently acquired first.

Note that when designing the SDTP, supporting file list requests containing ranges of values for *tags*, such as date ranges, was considered. However, because SDTP is focused on predetermined file transfers and not meant to be a more general subscription interface, *tag* values are limited to discrete values, not ranges.

3.4 File Transfer

The file is retrieved using a simple GET request containing the file identifier.

An example of a file GET request is:

```
curl   -X GET "https://provider.org/sdtp/v1/files/1342"
```

A successful HTTPS response is typically a status *Code 200 - Success* and the file contents. All responses include a *SDTP-TransactionID* in the header that is unique to each GET request. Responses for various error conditions are discussed in Section 4.

An alternative successful HTTPS response may be a status *Code 302 - File is available in another location* (aka. a redirect) with the header containing a *Location* field with the URL for the actual file content.

Note that the subscriber can transfer multiple files in parallel as long as the network and/or provider system are not overwhelmed. The provider/subscriber will typically agree on a maximum number of simultaneous file transfers (see Section 5 on Interface Parameters).

3.5 File Transfer Acknowledgment

For each successful file transfer, the subscriber performs a HTTPS DELETE containing the file's *fileid* to acknowledge the transfer completed successfully. At this point, the provider removes the file from the queue for the subscriber. Note, the purpose of this command is not for the provider to delete the file, but instead for the provider to remove the file entry from a file list queue for a specific subscriber.

Note that the subscriber should always check that the file size and checksum are correct before sending a successful file transfer acknowledgement (DELETE request).

An example of a file entry DELETE request is:

```
curl   -X DELETE "https://provider.org/sdtp/v1/files/1342"
```

A successful HTTPS response, returned after the provider removes the file from the queue, is *Code 204 - Success but no other response necessary*. All responses include a *SDTP-TransactionID* in

the header that is unique to each DELETE request. Responses for various error conditions are discussed in Section 4.

DELETE requests are idempotent, that is, the DELETE request can be issued multiple times and the results will be the same. This means that if a *fileid* is included in multiple DELETE requests, the return status will be Success each time. This also means that a DELETE request for a *fileid* that is not in the subscriber queue will return a success status (as long as the *fileid* is correctly formatted).

For current PDR protocol users, the DELETE request is similar to the *success* Product Acceptance Notice (PAN). There are no *failed* PAN or PDR Discrepancy (PDRD) equivalents because error handling is done out-of-band. Also, any reconciliation of transfers is handled out-of-band, similar to the current PDR protocol.

The SDTP also supports the ability to delete a range of file entries if the corresponding *fileids* are contiguous.

An example of a multi-file file entry DELETE request is:

```
curl   -X DELETE "https://provider.org/sdtp/v1/files/2061-2065"
```

In this example, any entries with *fileids* from 2061 to 2065 will be removed from the provider queue.

In the multi-file acknowledgement request, the subscriber should take care and be sure that all of the *fileids* in the range have been transferred successfully before performing the request. Note that the *fileids* assigned by the provider may not necessarily be contiguous. That is, the provider may provide the *fileids* like this: 3, 4, 5, but also like this: 3, 5, 7. Also, if *tags* are used by the subscriber to filter file entries, it is less likely that the *fileids* will be contiguous.

3.6 Extra Fields

Additional information about a file may be passed from the provider to the subscriber as values in the *extra* field. The contents of this field are negotiated between the provider and subscriber. One common use of this field is to group related files (see Section 3.7).
As with *tags*, when using *extra* fields that are metadata fields in the ESDIS Universal Metadata Model, a best practice is to use the UMM metadata field name (with the same capitalization), e.g. *ShortName*. Also, the valid values should be in-line with the UMM metadata model.

Some commonly used *extra* fields are similar to those mentioned in the *tags* fields, such as, *ShortName* and *Version*. In addition there may be other commonly used UMM fields that the provider/subscriber may agree upon, such as the file *Format*. It is possible to have the same name in the *tags* and *extra* fields, but this is not recommended.

3.7 Related File Groups

Files that are related to one another such as a browse or metadata file associated with a data file may need to be grouped together when received by the subscriber. Groups may also be used to

transfer multi-file granules. To group related files, all the files are assigned the same Group ID (a UUID) value by the provider. The group ID, the number of files in the group and the file number within the group (Table 3-3) are stored in a field named *group* within the *extra* field.

Table 3-3. Group Fields.

Name	Type	Description	Notes
groupid	string	Unique group identifier. Typically a UUID.	This is unique to this related file group and shall not be reused, e.g. *ba963192-04a5-11ea-9a9f-362b9e155667*
N	positive integer	Number of files within the group.	e.g. *6*
I	positive integer	Sequential file number within the group.	Range is from 1 to the number of files in the group (*n*). e.g. *1*

Example of group for a file within a file list:

```
{
  "files": [
    {
      "fileid": 1352,
      "name": "file1.txt",
      ...
      “extra”: {
        “group”: {“groupid”: “2a125b1a-4a7c-42e2-88f6-4fe19b172381”, “i”: 1, “n”: 6}
      }
    }, ...
  ]
}
```

Figure 3-2. Example group field.

The provider/subscriber pair may choose to use alternate methods to group related files, including:

- *taring* or *zipping* the a group of files and sending it as a single file;
- using a separate *group* file that contains a list of related files; or
- using a file naming convention that allows the subscriber to group the files implicitly.

3.8 Polling

The subscriber uses polling to obtain the next list of files to retrieve. Typically, the subscriber asks for a list of files, transfers each file and acknowledges each transfer, and then asks for another list. If a request for a new file list is made before the transfers are acknowledged for all of the files in the previous list, the unacknowledged file entries will again appear in the new list. So the subscriber should try to transfer all of the files in the current list before requesting a new list.

When the provider's file queue is empty, the subscriber will begin polling (asking for a new list of files) at a short time interval. This polling time interval is one that was agreed upon by the provider/subscriber pair for that *stream*. In some cases, the polling could continue for a long period

of time before new files are anticipated to be ready to be transferred. When this condition is detected by the subscriber, the subscriber can automatically increase the retry delay.

For nominal steady-state data flows with an appropriate polling time interval, the size of the file list is expected to be small. For instance, suppose the polling time interval was set to one minute. If the producer adds a few new files to the subscriber queue every minute and the subscriber is able to keep up with the flow by downloading the files quickly, then the queue will only contain a few files actively being downloaded by the subscriber and a few new files that have been added by the provider since the last list was obtain. However, the queue and corresponding file list could be much larger for a number of reasons, e.g. when the flow is starting up, the flow from the provider is intermittent (bursty), when a backlog accumulates because the subscriber can't keep up, after a subscriber and/or provider downtime, after network issues.

Note that this polling back-off approach is a best practice, but it is optional so it is something that is up to the individual provider/subscriber pair to decide whether to include. Also, an alternative approach to synchronous file list polling is to do the polling independently (asynchronously) by using a paging approach (next section).

3.9 File List Paging

The subscriber may want to page through the file list, so two optional query parameters are available for paging (Table 3-4).

Table 3-4. Pagination query parameters.

Name	Type	Description	Notes
maxfile	positive integer	Maximum number of file entries to return.	Default is the value agreed to by the provider and subscriber (see Section 5 on Interface Parameters).
startfileid	positive integer	*fileid* of the file entry prior to the first file entry to be returned.	e.g. *1352*

An example of a file list GET query with pagination is:

```
curl   -X GET "https://provider.org/sdtp/v1/files?stream=prod&ShortName=INS02A&
                 maxfile=10&startfieldid=1352"
                 -H "Accept: application/json"
```

As mentioned above, the provider orders the file entries in the subscriber queue by the time that the file entries were inserted into the queue. If the *startfileid* parameter is not given, the first file entry in the subscriber queue, after *tag* filtering, is returned.

For pagination, the *startfileid* is typically the *fileid* of the last entry in the previous file list. So, when the subscriber requests the next file list, the first *fileid* returned will be the one immediately after the last *fileid* returned in the previous request.

3.10 File Transfer Prioritization

To handle cases where there is a backlog, the provider/subscriber pair may need to agree on an approach to prioritize the order that the files are transferred.

One approach would be to use the *stream* tag to separate files with different priority into different subscriber queues. For instance, if there is a forward processing (*prod*) and a reprocessing (*reproc*) *stream*, then the subscriber could use the *stream* tag to pull the files from the *prod* stream first. This could be further refined by separating the prod stream into high and low priority streams, e.g. *prod_high* and *prod_low*.

Another approach would be for the subscriber to run several SDTP client instances, one for each *stream*, so that the files are processed independently. In this approach, the subscriber would still need an approach to adjust the pulling priorities of the different streams. This could be done by using an algorithm to allocate *connections* to the various subscriber client instances so that the higher priority files are transferred first.

3.11 Register Client Certificate

The subscriber may register a new client certificate with the provider using an HTTPS PUT request.

An example of a *register* PUT request is:

```
curl -X PUT "https://provider.org/sdtp/v1/register"
```

A successful HTTPS response, returned after the certificate is registered, is *Code 204 - Success but no other response necessary*. If the certificate provided is not from a certificate authority recognized by the provider, or if no certificate is provided, the error response is *Code 401 - Unauthorized*.

To prevent unwanted register requests, the provider may limit the times when client certificates can be registered. Outside of these times, a subscriber would receive the error response *Code 503 - Service Unavailable*.

The typical workflow for a new subscriber client certificate is as follows:

- Subscriber obtains a new client certificate from the agreed upon CA.
- Provider/subscriber agree on a time window to register the certificate.
- Provider enables the register end-point for the specified time period.
- During the time window, the subscriber performs a *register* request.
- Subscriber notifies the provider that a successful response was received from the *register* request.
- Provider accepts the certificate as valid, associates it with the appropriate subscriber streams, and notifies the subscriber.
- Subscriber does a *file list* request to verify that the connection is working correctly and notifies the provider.

4 ERROR HANDLING

The general philosophy for handling errors is that when most errors occur, there isn't anything that the provider's server software could automatically do to resolve the error. At any point, the provider can easily tell which files have not been transferred by examining a subscriber's queue and the subscriber has the same view of the files in the queue. Most errors require some manual intervention by the provider and/or subscriber to determine what went wrong. For common error scenarios (e.g. network errors, wrong product type, incorrect metadata, software bug), something needs to be fixed before the transfer can be retried. So the next step is always the provider/subscriber contacting each other to begin the debugging process.

When the interface is operating correctly, errors are expected to be rare and mainly due to network issues (authentication error, link down, etc.). A unique *SDTP-TransactionID* is provided in the header for each HTTPS request and so it can be used during communication to trace the error. A good practice is for the subscriber to maintain a log that contains information about each transaction, including the *SDTP-TransactionID*, to help with debugging any errors. The provider will also typically have a log of HTTPS requests that can be used to help understand the error.

The HTTPS response error codes returned by each transaction type are given in Table 4.

Table 4-2. HTTP error status responses.

Code	Description	Transaction type
400	The request is incorrect.	All
401	Request is not authenticated.	All
403	Request is authenticated but user is forbidden from accessing resource.	All
404	The requested resource does not exist.	File get and delete requests
429	Too many requests.	All

4.1 Incorrect File List

An incorrect file list may be detected by the subscriber. This could be because of a network transfer error, malformed json or incomplete file list. It also may be an issue or bug on the provider and/or subscriber side. For instance, the provider may provide a file not meant for a particular subscriber because of an operator error.

For intermittent network errors, the subscriber can retry the file list request several times. If still unsuccessful, the subscriber would work with the provider to resolve the network issue.

For other errors, the subscriber and provider need to contact each other and work together to resolve the issue.

4.2 File Transfer Error

The subscriber detects a file transfer error by checking a file's size and checksum. It may be caused by a network issue. It also may be an issue or a bug on the provider side, file corruption or other issue. The subscriber would typically retry to transfer a file several times. If the transfer does

not succeed after a number of tries, the subscriber can set that file aside and try other files in the list. If only a few files fail, the subscriber could continue to retrieve files.

The provider will remove files that have transferred successfully from the queue for the subscriber. When the subscriber sets a file aside, the provider will still include that file in the next file list requested by the subscriber. Note that in this case, it is possible for the transfer to stall if the number of files that have errors exceeds the maximum file list size.

Alternatively, the subscriber may decide to go ahead and acknowledge receipt of the file (using the *DELETE* response) and store the information related to the failed file transfer. The subscriber can then later work with the provider to resolve the file transfer error, and the provider can then put the file back into the queue for the subscriber.

There may be other errors unrelated to the file transfer, e.g. incorrect metadata, that may be detected after the file transfer occurs. In this case, the subscriber would ask the provider to correct the file and then the provider would add the corrected file to the subscriber's queue.

In many cases, the only way to resolve this issue is for the subscriber to contact the provider.

4.3 Related File Errors

There may be an error when all the related files in a group are not given in the file lists from the provider. There is no requirement that all related files for a group be in the same file list, but the expectation is that they will typically be in file lists that are temporally close to one another.

After waiting a sufficient time interval for all files with the same group *id* to become available, the subscriber would use the the number of files and the file sequence number to determine if any files are missing and if they are, communicate that information to the provider.

4.4 Incorrect Request (Code 400)

When the subscriber request is incorrect, the problem may be on the provider or subscriber side. The provider may have a bug in the provider interface, or the subscriber may be providing an invalid request. One possibility is that the subscriber is using a *tag* in a query that the provider has not implemented or the *tag* value is not valid. In either case, the provider and subscriber should contact each other and resolve the issue.

4.5 File Authentication Error (Code 401)

There is an issue with the authentication. The certificate may have expired or is not set up correctly. Subscriber should contact the provider.

4.6 Resource Error (Codes 403 and 404)

When the subscriber is forbidden from accessing a resource (Code 403) or the requested resource does not exist (Code 404), the problem may be on the provider or subscriber side. The provider may have a bug that prevents the resource from appearing in the provider interface, or the subscriber may be providing an incorrectly formatted resource identifier (e.g. non-numeric *fileid*). In either case, the provider and subscriber should contact each other and resolve the issue.

4.7 Too Many Requests (Code 429)

This error code is typically returned if the subscriber overloads the provider. When this occurs the subscriber should dial-back the number of requests to decrease the load on the provider. One mechanism to help prevent too many file downloads occurring simultaneously is for the subscriber to agree to limit the number of simultaneous downloads that will be performed (see Section 5 on Interface Parameters). If this error happens too often, the subscriber should build a mechanism to help prevent putting too much load on the provider. The subscriber could also discuss with the provider the need to increase the provider's capacity.

Check ESDIS CM library, https://ops1-cm.ems.eosdis.nasa.gov/cm2/, to verify that this is the correct version prior to use.

5 INTERFACE PARAMETERS

There are a number of parameters that control (tune) the interface. The provider and subscriber will agree on values or ranges of values for these parameters.

Parameter values may be *stream* specific. For example, a test *stream* may have different values than a production *stream*.

Table 5-3. Interface control parameters and their default values.

Parameter	Default Value(s)
Maximum number of files in list	10000
Number of empty polls before retry interval increases	3
Short polling retry interval (seconds)	1
Medium polling retry delay interval (seconds)	300
Long polling retry delay interval (seconds)	3600
Number of retries for file content transfer error	3
Checksum types	md5, sha256
Maximum number of simultaneous download threads	5
Default expiration time period (days)	180

Note that the checksum is used for message integrity, i.e. to check for transfer errors. The checksum must not be used for cryptographic purposes, such as a digital signature, or as a secure hash.

As mentioned above, when the interface is first set up and as it evolves over time, the provider/subscriber pair must agree on additional interface information. This includes: provider URL, points of contact, subscriber certificate DN, *tags* and their valid values, *extra* fields and their valid values, and file grouping approach (if needed).

Check ESDIS CM library, https://ops1-cm.ems.eosdis.nasa.gov/cm2/, to verify that this is the correct version prior to use.

6 RUNNING THE PDR PROTOCOL ON TOP OF SDTP

This section describes how a data provider and subscriber can run the existing PDR protocol on top of SDTP. Note that this is not the preferred approach, but is something that legacy PDR protocol users may choose to do.

In this intermediate approach the provider/subscriber will still pass the data files and the PDR, PAN and PDRD files, but via the HTTPS protocol instead of the SFTP protocol. To do this, both the provider and subscriber will set up both an SDTP server and client. The provider SDTP server has a queue for the subscriber with the PDR and data files. The subscriber's SDTP client transfers those files from the queue via HTTPS. The subscriber SDTP server has a queue for the provider with the PAN and PDRD files. The provider SDTP client transfers all of the files from the queue via HTTPS.

Basically each side designates an incoming and an outgoing directory and runs a SDTP server. The outgoing directory gets mounted as a volume to the nginx container to serve every file in it. Each side runs two daemons. One monitors the outgoing directory and PUTs any new files to its own server with a single tag designating the subscriber of the other side, which has a subscription to get every file with that *tag*. The second daemon is the polling downloader that connects to the other side's SDTP server and just asks for a new filelist periodically and downloads any files it sees to the site's own incoming directory.

Existing PDR protocol software can just watch the incoming directory for the data files and PDR files, etc., and does the normal thing, putting PAN files, PDRD files, etc., in the outgoing directory which the other side will then see in their incoming directory.

In addition, the *stream* tag could be used by both sides to separate the various test, forward and reprocessing flows into different directories on the provider and subscriber side.

Internally, a site could even use SFTP/SCP (or other approaches) to get to a new SDTP server host, so it would be almost seamless to plug into the existing systems. In this case, there would also be no need to modify the provider and/or subscriber databases to support SDTP, i.e. they can each use a secondary SDTP server database that only contains file transfer information.

This doesn't change SDTP, it is just a way to enable existing PDR protocol users a path to migrate their existing systems.

Appendix A Abbreviations and Acronyms

CA	Certificate Authority
CCB	Configuration Change Board
CCR	Configuration Control Request
CMO	Configuration Management Office
CNM	Cloud Notification Mechanism
CSV	Comma-Separated Values
DN	Distinguished Name
DAAC	Distributed Active Archive Center
ECS	EOSDIS Core System
EOS	Earth Observing System
EOSDIS	Earth Observation System Data and Information System
EDOS	EOS Data and Operations System
ESDT	Earth Science Data Type
ESDIS	Earth Science Data and Information System
GMT	Greenwich Mean Time
GSFC	Goddard Space Flight Center
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over TLS
FTP	File Transfer Protocol
ICD	Interface Control Document
ID	Identifier
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
L1	Level 1
NAMS	NASA Access Management System
NASA	National Aeronautics and Space Administration
OA	Operations Agreement
PAN	Product Acceptance Notice
PDR (1)	Polling with Delivery Record (a protocol)
PDR (2)	Product Delivery Record (a file)
PDRD	PDR Discrepancy
SDS	Science Data System
SDTP	Science Data Transfer Protocol
SFTP	Secure File Transfer Protocol
SIPS	Science Investigator-led Processing System
TLS	Transport Layer Security
URL	Universal Resource Locator
UMM	Universal Metadata Model
UUID	Universally Unique Identifier
VIIRS	Visible Infrared Imaging Radiometer Suite