Data Router

Provisioning API

Background

The Data Router (DR) provisioning API is an HTTPS-based, REST-like API for creating and managing DR feeds and subscriptions.

The DR provisioning API is not meant to be used directly by DR end users (publishers and subscribers). Instead, prospective publishers and subscribers will use a Web-based interface to a metadata repository that houses information about various data sources in the corporation, including DR data feeds. The metadata repository will call the DR provisioning API, as needed, to manage feeds and subscriptions. In this document, the term *client system* (or, more briefly, *client*) refers to a system that accesses the provisioning API directly. Currently the only client system is the Enterprise Metadata Repository system (EMR), known as DPLR.

Note on terminology: In some high-level descriptions of the DR, there is a notion that a feed can have multiple versions. A new version typically indicates that the payload structure or per-file metadata schema for the feed has changed in some way. In this document, the term "feed" refers to a single version of a feed.

Changes to the Provisioning API in Version 2.1

Version 2.1of the API will be introduced as part of DR Release 3 (R3).

DR introduced a new field called business description to the feed creation. The existing feed description has been renamed to technical description. Created date field is added for the subscription and feed. A new logging framework is introduced called Ecomp Logging framework . Two log files have been added to this following the EELF standards. Groups feature is added to leverage similar privileges as primary owner to update Feed/Subscriptions in absence of owner. This gives the ability to act upon feeds published/subscribed by other team members in order to provide business continuity.

Identity, Authentication, and Authorization

The DR provisioning system uses TLS certificates to identify and authenticate clients of the provisioning API. A client must present a valid certificate, signed by an authority recognized by the DR, with a subject

that has been registered with the DR provisioning system. In addition, the provisioning system verifies that the source IP address of an incoming provisioning request appears on a list of authorized IP addresses that have been registered with the DR provisioning system. All incoming provisioning requests are subjected to this identification, authentication, and authorization process.

Responsibility for identification, authentication, and authorization at the level of individual users is divided between DPLR, working in conjunction with the Policy Engine (PE), and the DR provisioning system. Specifically:

- DPLR is responsible for identifying and authenticating a user with the AT&T Global Logon system, with the user's ATTUID acting as the user identity. DPLR includes the ATTUID of the user initiating any provisioning activity in the `X-ATT-DR-ON-BEHALF-OF` header in each HTTP request it makes to the Provisioning API. The DR provisioning system relies on DPLR to perform the necessary authentication.
- DPLR consults the PE to decide whether to allow a user to create a feed or to create a subscription to an existing feed. If a feed or subscription creation operation is not authorized, DPLR does not send a request to the Provisioning API. The DR provisioning system relies on DPLR and the PE to perform authorization and will accept any valid feed or subscription creation request from DPLR.
- The DR provisioning system performs some simple user-level authorization of other provisioning requests[1]:
    - The DR provisioning system allows only the user who created a feed or subscription to retrieve information about that feed or subscription. (Specifically, the ATTUID passed in a request to retrieve information about a feed or subscription must match the ATTUID that was passed in the request that created the feed or subscription.)
    - The DR provisioning system allows only the user who created a feed or subscription to modify or delete that feed or subscription. (Specifically, the ATTUID passed in a request to modify or delete a feed or subscription must match the ATTUID that was passed in the request that created the feed or subscription.)
    - The DR provisioning system allows only the user who created a subscription to reset the retry mechanism for that subscription. (Specifically, the ATTUID passed in a request to reset the retry mechanism for a subscription must match the ATTUID that was passed in the request that created the subscription.)

[1] This authorization functionality in the DR provisioning subsystem comes from DR R1, when there was no PE. In a future release, this functionality may be moved to DPLR and the PE.

API Specification

The API Provisioning Model

The DR provisioning API defines two resource types—the *feed* and the *subscription*, each with JSON representations. The API models the provisioning data as a collection of feeds that are known to the DR (the *feeds collection*), with each feed containing a collection of the subscriptions to the feed. The standard HTTP operations (POST, GET, PUT, and DELETE), used in conjunction with these resource representations, allow an API user to create, get information about, modify, and delete feeds and subscriptions.

The API exposes URLs that a client system uses when making API requests. These URLs address resources and collections of resources. The API is designed so that a client system needs to know only one URL in advance in order to get started with provisioning, namely, the URL that points to the feeds collection[2]. All of the other URLs that a client system needs are generated by the DR when the DR creates feeds and subscriptions. These URLs are returned to the DR in the DR's responses to API requests. This document uses symbolic names, rather than literal URL strings, for the various URLs. The URL for the feeds collection is given the symbolic name <*drFeedsURL*>. Later sections of the document will introduce the other URLs and explain how they are used.

Common Characteristics of API Requests

A client system invokes the DR provisioning API by making HTTP requests to appropriate URLs, as described in more detail below. All API HTTP requests share some common characteristics:

- All API requests are secured using HTTPS, to provide for encryption and for mutual authentication of the client system and the API server.
    - The DR uses a server certificate signed by VeriSign and issued under AT&T Services, Inc.[3] A client system must be configured to recognize this certificate authority. As is customary in HTTPS, the common name in the server certificate's subject will match the fully-qualified domain name used to reach the server hosting the API.
    - The DR requires the client system to present a client certificate signed by VeriSign and issued under AT&T Services, Inc[4]. The DR maintains (through a mechanism outside the scope of this document) a list of client subjects authorized to use the API. The certificate presented by a client must have a subject on the list of authorized subjects. Prior to using the provisioning API, the operators of a client system must provide the DR tier 3 field support team a list of the subjects it will use in its client certificates.

[2] In the current production environment, this URL is `https://data-router.com/`.

[3]

[4]

- All API requests must originate from a list of authorized source IP addresses. The DR maintains (through a mechanism outside the scope of this document) a list of IP addresses from which API requests are permitted. Prior to using the provisioning API, the operators of a client system must provide the DR tier 3 field support team a list of the IP addresses from which requests will be sent.
- All API requests must contain an `X-ATT-DR-ON-BEHALF-OF` header containing the ATTUID of the publisher or subscriber on whose behalf the request is being made. In making a request, the client system asserts that the identity has been authenticated. The value in this header must be 8 characters long, or shorter. The API server truncates longer values.
- All API requests that include a body must include a `Content-Type` header whose value accurately reflects the type of content being provided, using one of the DR-specific media types defined below.
- The response from the API server to any API request will indicate the outcome of the request using one of the standard HTTP status codes[5]. The exact codes used depend on the type of request and are described later.

Feed Management

Creating a new feed means adding a feed resource to the DR's feeds collection by POSTing a representation of the new feed resource to the URL representing the collection, *<drFeedsURL>*.

When the DR creates a feed in response to an API request, the DR assigns a unique URL for the feed. This document uses the symbolic name *<feedURL>* to refer to an instance of such a URL. The client system uses the feed URL to make changes to the feed, to delete the feed, or to retrieve information about the feed. The DR also assigns distinct URLs used for publishing to a feed (symbolic name *<publishURL>*), subscribing to a feed (symbolic name *<subscribeURL>*), and obtaining log information related to activity on the feed (symbolic name *<feedLogURL>*).

Feed Representations

The API uses JSON objects to represent feeds. The API supports two representations of a feed:

- The full representation, including all of the information that the DR holds about a feed. This representation has the media type: `application/vnd.att-dr.feed-full;version=2.0`[6].

[5] See RFC 2616, section 10, for definitions of all the standard status codes.

[6] The API uses a `version` parameter on each of the media types it defines. This makes it easier to change representations in the future while maintaining backward compatibility. If the `version` parameter is omitted, the DR will assume a value of `2.0`. (Version `1.0`, used in DR R1, does not include the `suspend` field.) In DR R2 and beyond, the Provisioning API will accept requests containing version 1.0 or version 2.0 objects. (In the case

- A representation that contains just the information about a feed that is set by the client system. This representation has the media type: `application/vnd.att-dr.feed;version=2.0`.

Table 1 below shows the fields in the full representation of a feed, listing restrictions on the content of each field as well as indicating what entity sets the value of the field (the client system or the DR) and whether the value can be updated after the feed has been created. Table 2 and Table 4 provide additional details.

Table 1 : Feed Object

Field	Type	Description	Restrictions	Set By	Updatable?	Required
name	string	Feed name	Length <=20	Client	No	Yes
Version	String	Feed version	Length <= 20 Name/version Combination must be unique in the DR	Client	No	Yes
Description	String	Feed Description	Length <= 256	Client	Yes	No
Business Description	String	Business Description	Length <=256	Client	Yes	No
Authorization	Object	Information for authorizing publishing requests -		Client	Yes	Yes
Suspend	Boolean	Set to `true` if feed is in suspended state		Client	Yes	No; if absent defaults to `false`
Publisher	String	Publisher identity as passed in X-ATT-DR-ON-BEHALF-OF at creation time		DR	No	Yes

of a version 1.0 object, the Provisioning API will use the default value (`false`) for the absent `suspend` field.) The Provisioning API will always return a version 2.0 object in a response.

Links	Object	URLs related to this feed - see Table 4 below		DR	No	Yes
groupid	Integer			client	yes	Yes

The `authorization` field is a JSON object containing lists of the endpoint addresses and identifiers that are authorized to publish to the feed. All fields are required, set by the client, and updatable. Table 2 describes the object's fields.

Table 2: Feed Authorization Object

Field	Type	Description	Restrictions
`classification`	string	An indicator of the feed's data security classification	Length <=32
`endpoint_ids`	object[]	Array of objects defining the identities that are allowed to publish to this feed—see Table 3 below	At least 1 id in the array
`endpoint_addrs`	string[]	Array of IP addresses or IP subnetwork addresses that are allowed to publish to this feed; an empty array indicates that publish requests are permitted from any IP address	Each string must be a valid textual representation of IPv4 or IPv6 host address or subnetwork address.

[7] The name and version, taken together, uniquely identify the feed.

The `endpoint_ids` array contains objects that define an identifier and password that a publishing endpoint will use to authenticate to the DR publishing API. Table 3 lists the fields in an endpoint identity object.

Table 3: Endpoint Identity Object

Field	Type	Description	Restrictions

id	string	Publishing endpoint identifier	Length <= 20
password	string	Password associated with id	Length <= 32

The links field in the authorization object is a JSON object containing the links (URLs) that the DR has generated for the feed. These links are used for various operations involving the feed, such as publishing files to the feed and subscribing to the feed. Table 4 lists the object's fields, all of which are set by the DR and none of which can be updated. The Symbolic Name column gives the name that this document uses to refer to instances of these links.

Table 4: Feed Links Object

Field	Description	Symbolic Name
self	URL pointing to this feed, used for updating and deleting the feed	*<feedURL>*
publish	URL for publishing requests for this feed	*<publishURL>*
subscribe	URL for subscribing to this feed	*<subscribeURL>*
log	URL for accessing log information about this feed	*<feedLogURL>*

The application/vnd.att-dr.feed representation has the same fields as the full representation, except that it does not include the publisher and links fields.

Figure 1 below shows an example of a JSON object containing the full representation of a feed resource.

```
{
    "name" : "feedx",
    "version" : "v1.0.0",
    "description" : "My example feed",
    "business_description" : " This is a feed for bus_desc",
    "authorization" :
        {
            "classification" : "unrestricted",
            "endpoint_ids" : [
                {"id" : "pub01", "password" : "relkwelj"},
                 {"id" : "pub06", "password" : "o9eq1mbd"}
            ],
            "endpoint_addrs" : ["10.0.0.1", "192.168.0.1", "10.10.10.0/24"]
        },
    "suspend" : false,
    "publisher" : "pub393",
    "links" :
        {
            "self" : "https://feeds.dr.att.com/6YeW23nZ",
```

```
            "publish" : "https://pub.dr.com/RsgI1c6x", "
            subscribe" : "https://sub.dr.com/eM2q4At9", "log
            " : "https://logs.dr.com/yVqw90tF"
        }

    "created_date" : "'2016-06-29 11:26:11'"      ,
    "Last_Modified" : "2016-07-04" 04:15:15'"      ,
    "groupid" : "22" ,
    "changeowner" : true,

}
```

Figure 1: Example of a Feed Object
The identifiers and URLs used in the production system may be quite different from the ones shown here.

Creating a Feed

To create a feed, a client system POSTs an `application/vnd.att-dr.feed` representation of the feed to the *<drFeedsURL>*. The POST request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 201 ("Created") and a body containing the complete representation of the newly-created feed (`application/vnd.attdr.feed-full`). The response will also include a `Location` header containing the URL of the newly-created feed (*<feedURL>*)[7].

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Modifying a Feed

To modify an existing feed, a client system makes an HTTP PUT request to the *<feedURL>*, with the body containing an `application/vnd.att-dr.feed` representation of the modified feed. The PUT request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 200 ("OK") and a body containing the complete representation of the modified feed (`application/vnd.att-dr.feedfull`).

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Suspending or Reinstating a Feed

Suspending or reinstating a feed is just a special case of modifying a feed. To suspend a feed, the representation of the modified feed includes the `suspend` parameter with a value of `true`. To

[7] This same URL is also carried in the `self` field of the `links` object in the full representation of the feed.

reinstate a feed, the representation of the modified feed includes the `suspend` parameter with a value of `false`.

Deleting a Feed

To delete a feed, a client system makes an HTTP DELETE request to the *<feedURL>*. The DELETE request must conform to the common characteristics of API HTTP requests, described above. The request has no body.

If the request is successful, the DR will return a response with a status code of 204 ("No Content"). The response will have no body.

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Note that it is not necessary to suspend a feed prior to deleting it.

Retrieving Information about a Feed

To retrieve a representation of a feed, a client system makes an HTTP GET request to the *<feedURL>.* The GET request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 200 ("OK") and a body containing the complete representation of the feed (`application/vnd.att-dr.feed-full`).

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Subscription Management

Every DR subscription is associated with a specific feed—the feed from which the subscription will receive published files. In the DR provisioning API's resource model, each feed has a collection containing all of the subscriptions to that feed. This collection is identified by a URL, *<subscribeURL>*, generated by the DR when the feed is created and returned to the client system in the response to the API request that created the feed. A client system creates a subscription to a feed by POSTing a representation of the new subscription resource to the feed's *<subscribeURL>*.

When the DR creates a subscription in response to an API request, the DR assigns the subscription a unique subscription URL (symbolic name *<subscriptionURL>*). The client system uses the subscription URL to make changes to the subscription, to delete the subscription, or to retrieve information about the subscription. The DR also assigns a distinct URL used for obtaining log information related to activity on the subscription (symbolic name *<subLogURL>*).

Subscription Representations

The API uses JSON objects to represent feeds. The API supports two representations of a subscription:

- The full representation, including all of the information that the DR holds about a subscription. This representation has the media type: `application/vnd.att-dr.subscription-full;version=2.0`[8].
- A representation that contains just the information about a subscription that is set by the client system. This representation has the media type: `application/vnd.att-dr.subscription;version=2.0`.

Table 5 below shows the fields in the full representation of a subscription, listing restrictions on the content of each field as well as indicating what entity sets the value of the field (the API client or the DR) and whether the value can be updated after the subscription has been created. Table 6 and Table 7 provide additional details.

Table 5: Subscription Object

Field	Type	Description	Restrictions	Set By	Updatable?	Required?
`delivery`	object	Address and credentials for delivery — see Table 6 below		Client	Yes	Yes
`follow_redirect`	boolean	Set to true if feed redirection is expected		Client	Yes	Yes
`metadataOnly`	boolean	Set to `true` if subscription is to receive only per-file metadata		Client	Yes	Yes
`suspend`	boolean	Set to `true` if the subscription is in the suspended state		Client	Yes	No; if absent, defaults to `false`

[8] As with the feed media types, if the `version` parameter is omitted, the DR will assume a value of `2.0`. (Version `1.0`, used in DR R1, does not include the `suspend` field.) In DR R2 and beyond, the Provisioning API will accept requests containing version 1.0 or version 2.0 objects. (In the case of a version 1.0 object, the Provisioning API will use the default value (`false`) for the absent `suspend` field.) The Provisioning API will always return a version 2.0 object in a response.

`subscriber`	string	Subscriber identity, as passed in `X-ATT-DR-ONBEHALF-OF` at creation time		DR	No	Yes
`links`	object	URLs related to this subscription—see Table 7 below		DR	No	Yes
`groupid`	Integer			Client	Yes	Yes

The `delivery` field is a JSON object containing the information that the DR will need to deliver files to the subscriber. Table 6 describes the object's fields, all of which are required, set by the client, and updatable.

Table 6: Delivery Object

Type	Description	Restrictions
string	URL to which deliveries for this subscription should be directed	Valid HTTPS URL, length <= 256
string	User ID to be passed in the Authorization header when deliveries are made	Length <= 20
string	Password to be passed in the Authorization header when deliveries are made	Length <= 32
boolean	Flag indicating whether the DR should use the HTTP 100-continue feature	Must be: • `true` to use 100continue • `false` to disable using 100-continue

The `links` field is a JSON object containing the links (URLs) that the DR has generated for the subscription. Table 7 describes the object's fields, all of which are set by the DR and none of which can be updated.

Table 7: Subscription Links Object

Field	Description	Symbolic Name
`self`	URL pointing to this subscription, used for updating and deleting the subscription	*<subscriptionURL>*

`feed`	URL of the feed to which this subscription applies; the same URL as the <feedURL> in the representation of the feed	*<feedURL>*
`log`	URL for accessing log information about this subscription	*<subLogURL>*

```
{
   "delivery" :
        {
            "url" : "https://sub.example.com/myfeed", "
            user" : "datarouter",
            "password" : "rewqpoiu",
            "use100" : true         },
    "metadataOnly" : false,
    "follow_redirect" : false,
    "suspend" : false,
    "subscriber" : "sub949",
    "groupid" : "22",
    "links" :  {
            "self" : "https://subs.dr.com/kk394Wzz",
                "feed" : "https://feeds.dr.com/6YeW23nZ",
            "log" : "https://logs.dr.com/Ui2XbnaG" }

}
```

Figure 2: Example of a Subscription Object
The identifiers and URLs used in the production system may be quite different from the ones shown here

Figure 2 above gives an example of a JSON object containing the full representation of a subscription resource.

The `application/vnd.att-dr.subscription` representation has the same fields as the full representation, except that it does not include the `subscriber` and `links` fields.

Creating a Subscription

To create a subscription, a client system POSTs an `application/vnd.att-dr.subscription` representation of the subscription to the *<subscribeURL>* for the feed. The POST request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 201 ("Created") and a body containing the complete representation of the newly-created subscription (`application/vnd.att-dr.subscription-full`). The response will also include a `Location` header containing the URL of the newly-created subscription (*<subscriptionURL>*)[9].

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Modifying a Subscription

To modify an existing subscription, a client system makes an HTTP PUT request to the *<subscriptionURL>*, with the body containing an `application/vnd.att-dr.subscription` representation of the modified subscription. The PUT request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 200 ("OK") and a body containing the complete representation of the modified subscription (`application/vnd.attdr.subscription-full`).

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

[9] This same URL is also carried in the `self` field of the `links` object in the full representation of the subscription.

Suspending or Reinstating a Subscription

Suspending or reinstating a subscription is just a special case of modifying a subscription. To suspend a subscription, the representation of the modified subscription includes the suspend parameter with a value of true. To reinstate a subscription, the representation of the modified subscription includes the suspend parameter with a value of false.

Deleting a Subscription

To delete a subscription, a client system makes an HTTP DELETE request to the *<subscriptionURL>*. The DELETE request must conform to the common characteristics of API HTTP requests, described above. The request has no body.

If the request is successful, the DR will return a response with a status code of 204 ("No Content"). The response will have no body.

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Note that it is not necessary to suspend a subscription prior to deleting it.

Retrieving Information about a Subscription

To retrieve a representation of a subscription, a client system makes an HTTP GET request to the *<subscriptionURL>.* The GET request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 200 ("OK") and a body containing the complete representation of the subscription (`application/vnd.attdr.subscription-full`).

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Resetting a Subscription's Retry Schedule

When the DR is unable to make a delivery to a subscription, under certain circumstances the DR will hold the file and retry the delivery according to a retry schedule. In some cases, a subscriber will fix whatever problem was causing the delivery failures and wants the DR to make a new attempt immediately, bypassing the retry schedule. The provisioning server will accept a request to reset the retry schedule and will then signal the DR nodes to begin delivering any files for that subscription as soon as the node has delivery capacity available.

To make a request to reset a subscription's retry schedule, the client makes an HTTP POST request to the *<subscriptionURL>.* The POST request must conform to the common characteristics of API HTTP requests, described above. The POST request has a body with a content type of

`application/vnd.att-dr.subscription-control`. The body is a JSON object with a single boolean property, `failed`. To reset the retry schedule (by indicating that the subscription is no longer in a failed delivery state), this property is set to `false`[10]. (The provisioning server will accept a request with the `failed` property set to `true`, but the request will have no effect.)

Figure 3 below shows an example of the subscription control object used to reset the retry schedule.

```
{
    "failed" : false
}
```

Figure 3: Example of a Subscription Control Object

If the request is successful, the DR will return a response with a status code of 202 ("Accepted"). The response will have no body. Note that a success response means only that the DR provisioning server recognizes the request as a valid one and will attempt to signal the DR nodes to begin deliveries for the subscription. A success response does not mean that deliveries have actually started. Note also that in some cases, a DR node may be out of service and thus will not receive the request. If such a node holds deliveries for the subscription, that node will not begin deliveries for the subscription until it comes back into service.

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Feeds Collection Queries

The DR provisioning API supports queries asking about the feeds known to the DR. A client system makes such a query by directing a GET request to the *<drFeedsURL>*, optionally including a query string, as described in Table 8. The GET request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 200 ("OK") and a body containing the result. As Table 8 indicates, some queries return a full representation of a single feed. Responses to these queries return a body with `application/vnd.att-dr.feed-full` content. Other queries return a list of *<feedURL>* values. The bodies for responses to such queries have a `Content-Type` of `application/vnd.att-dr.feed-list`. The body consists of a JSON array of strings, with each string being a *<feedURL>*.

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

[10] The rationale for naming the field "failed" is that the DR considers a subscription that's awaiting a retry after a delivery failure to be in a "failed" state. The POST request sets the subscription's state back to a non-failed state, so that the DR will attempt to deliver to it as soon as it can, rather than waiting for the retry interval to expire.

Table 8: Feeds Collection Queries

Query String	Response
(none)	List of the *<feedURL>*s for all of the feeds known to the DR.
?name=*feed_name*	List of the *<feedURL>*s for all of the feeds whose `name` fields have the value *feed_name*.
?name=*feed_name*&version=*version_id*	Full representation of the feed whose `name` field has the value *feed_name* and whose version field has the value *version_id*.
?publisher=*publisher_identity*	List of the *<feedURL>*s for all of the feeds whose publisher fields have the value *publisher_identity*.
?subscriber=*subscriber_identity*	List of the *<feedURL>*s for all of the feeds to which a subscriber with identity *subscriber_identity* has a subscription.

Subscription Collection Query

Each feed representation points to a collection containing the subscriptions to the feed. This collection is addressed using the *<subscribeURL>* for a feed. A client system can obtain a list of the *<subscriptionURL>*s for all of the subscriptions to the feed by making an HTTP GET operation against the *<subscribeURL>*. The GET request must conform to the common characteristics of API HTTP requests, described above.

If the request is successful, the DR will return a response with a status code of 200 ("OK") and a body with a Content-`Type` of `application/vnd.att-dr.subscription-list`. The body consists of a JSON array of strings, with each string being a *<subscriptionURL>*.

If the request fails, the DR will return a response with a status code indicating an error. See Appendix 1 for a description of common error cases.

Business Description

A new field is added to the Data router during the creation of the feed called Business Description. The purpose of this field is to provide the description related to the business aspects of the feed. This field is not mandatory.

The below figure shows the JSON example for business description in a feed

```
{
    "business_description" : "Test Business description."
}
```

Created date

This is a new addition to the feed creation. Whenever a new feed is created , the timestamp of the feed is recorded is the backend. These timestamps will be updated every time whenever a change has been made to the feed. These updated feed will be tagged as Last modified and the timestamp while creating the feed will be created date.

```
created_date : 2016-06-29 11:26:11
Last_Modified :  2016-07-04 04:15:15
```

Groups

Groups are new features added to the data router during the creation of the feed. When a feed is created, an option is provided to select the group functionality. The members of that individual group will have access to the feed details and the authorization to edit/delete that feed. Similar to the individual users feed creations, the feed created with group functionality will have an unique URL for the feed assigned by the data router. However, this group feature is optional.

A representation that contains just the information about a group that is set by the client system. This representation has the media type: `application/vnd.att-dr.groups;version=2.1`

Feed/Subscription Groups

Field	Type	Description	Length
authid	String	Authentication ID sent by DPLR	50
name	String	Name of the Group	Length<=20
description	String	Description of the group	
classification	String	An indicator of the feed's data security classification	Length <=32

members	String	Members who belong to the group	

Add/Edit Groups

Adding or editing a group can be performed by two ways. Firstly, by using a json object and secondly by using the curl command. Listed below are some examples of the json object.

JSON object to add or Edit the group

```
{
                "authid": "GROUP-0000-c2754bb7-92ef-4869-9c6b-1bc1283be4c0",
                "name": "Test Group",
                "description": "Test Description of Group .",
                "classification": "publisher/subscriber",
                "members": "{id=attuid, name=User1}, {id=attuid, name=User 2]"
}
```

Request for Co-owner of the group

When a feed is created by the user from a group , the members from the group have to raise a request to be the co-owner of the feed. (i.e) everyone in the group can access the feed in order to edit or modify the feed or subscription.

Example for an edit request as a co-owner for the feed

```
-H "X-ATT-DR-ON-BEHALF-OF: ATTUID"  -H "X-ATT-DR-ON-BEHALF-OF-GROUP: GROUP-0000-c2754bb7-92ef-4869-9c6b-1bc1283be4c0
```

Change ownership of the feed

The JSON PUT will have a parameter for the change of owner for the feed. It will change the ownership only if "changeowner = **`true`** " with group header parameter (e.g. "X-ATT-DR-ON-BEHALF-OF-GROUP: GROUP-0000-c2754bb7-92ef-4869-9c6b-1bc1283be4c0"). If the group header parameter is missing, it would be ignored.

```
{
        "changeowner" : true/false,
}
```

Appendix 1: Error Conditions

The DR provisioning API uses standard HTTP error status codes to indicate problems in handling a request. Table 9 lists the codes that a client system is most likely to encounter while using the API. This is not an exhaustive list—other codes may be used, and some HTTP libraries that a client system might use sometimes generate error responses locally, outside of the control of the API.

In keeping with HTTP's standard usage of status codes, codes in the range 400-499 indicate a problem with the request. Requests that elicit a response in the 400-499 range should not be retried without first being modified to correct the problem. Status codes in the 500-599 range indicate a problem on the server side. Requests that elicit a response in the 500-599 range can be retried once the server problem is cleared.

The DR provisioning API never redirects requests and so does not use status codes in the range 300-399.

A response with an error status code may include a body with HTML or plain text content providing a description of the problem.

Table 9: Error Status Codes

Code	Meaning	Description
400	Bad Request	The request is defective in some way. Possible causes: • JSON object in request body does not conform to the spec. • `X-ATT-DR-ON-BEHALF-OF` header missing.
401		Indicates that the request was missing the Authorization header or, if the header was presented, the credentials were not acceptable
403	Forbidden	The request failed authorization. Possible causes: • Request originated from an unauthorized IP address. • Client certificate subject is not on the API's authorized list. • `X-ATT-DR-ON-BEHALF-OF` identity is not authorized to perform the requested action.
404	Not Found	The Request-URI does not point to a resource that is known to the API.
405	Method Not Allowed	The HTTP method in the request is not supported for the resource addressed by the Request-URI.
415	Unsupported Media Type	The media type in the request's `Content-Type` header is not appropriate for the request.
500	Internal Server Error	The DR API server encountered an internal error and could not complete the request.
503	Service Unavailable	The DR API service is temporarily unavailable. The response may include a `Retry-After` header indicating when the client system should retry the request.
200 to 299		Success Response
-1		Fail Delivery

Change History

Version 2.1

Changes to the DR including the introduction of Groups feature , addition of the field business description, and created date.

Version 2.0.1

Changes for DR R3, including the introduction of a capability to reset a subscription's retry timer.

Version 2.0

Changes for DR R2, including the introduction of the `suspend` field in the feed and subscription objects.

Version 1.4

Passwords are not obscured in the response to GET requests against feeds and subscriptions. (Bug #30) Added language to clarify that the `endpoint_addrs` array may be empty. (Bug #28) Remove language stating that the feeds collection URL has not yet been defined, and add a footnote giving the current production URL.

Version 1.3

Document length restriction on the X-ATT-DR-ON-BEHALF-OF header value. (Bug #11)

Version 1.2

Add information about certificate signing .

Add the `version` parameter to media types.

Add `use100` flag to the `delivery` object for subscriptions, to control the DR's use of the HTTP 100continue feature when making a delivery.

Version 1.1

Correct the name of the HTTP header ("Authorization", not "Authentication") in Table 6. Correct typos.

Version 1.0

Initial version; reviewed on March 8, 2013 and base lined with no substantive changes.