

**Subscription Management API**

Robert J. Muller

August 6, 2015

643 Bair Island Road Suite 403  
Redwood City, CA 94063

The API calls are URIs that identify resources or actions. You can use the GET, POST, PUT, and DELETE HTTP operations to query, create, update, or delete the resources, respectively. The API calls have comments (#) that describe the resource or action. The substitution value <id> is a unique integer identifier for an object, used throughout the documentation. You submit new and updated data values in the body of the POST or PUT operations. All results are either JSON or HTML. All JSON results are JSON arrays, though the array may contain only a single object.

The documentation conventions here show the HTTP operation, the base URI without the domain https://api.arabidopsis.org, the possible URI parameters that can appear in the query string following a "?", and the type of the returned data. For example, the operation to retrieve all partners:

GET /partners

* returns JSON array of all partners

[   
 {"partnerId": "tair","name": "TAIR",  
 "logoUri": "<https://s3-us-west-2.amazonaws.com/pw2-logo/logo2.gif>",  
 "termOfServiceUri": "<https://www.arabidopsis.org/doc/about/tair_terms_of_use/417>"  
 },  
 {"partnerId": "yfd", "name": "YFD",   
 "logoUri": "<https://s3-us-west-2.amazonaws.com/pw2-logo/yfd.png>",  
 "termOfServiceUri": "<https://www.google.com/intl/en/policies/terms/?fg=1>"  
 }  
]

The documentation shows the URI (the full URI would be, for example, "https://api.arabidopsis.org/partners") and documents the nature of the returned data with an example of the returned JSON. The HTTP return code is 200 (OK) unless stated otherwise.='api

The operation to get the subscriptions active for a particular partner:

GET /subscriptions/active?partnerId=<id>

* where <id> is a partner id
* returns JSON array of subscriptions

[  
 {"subscriptionId": 1, "partyId": 10, "partnerId": "tair",   
 "startDate": "2014-04-01T12:00:00Z", "endDate": "2017-03-31T12:00:00Z"  
 },  
 {"subscriptionId": 2, "partyId": 11, "partnerId": "tair",   
 "startDate": "2014-04-01T12:00:00Z", "endDate": "2017-03-31T12:00:00Z"  
 }  
]

This example shows a parameter, partnerId, and it's value and documents the nature of that value, then shows the nature of the returned data with an example.

# Partners

A *partner* is a system that comprises a unit for subscription, authentication, and authorization. The user accesses the partner system through a well-defined set of uniform resource identifiers (URIs) identified by a set of regular-expression patterns. Partners have individual subscription terms for display on subscription pages. Terms have period, price, and group discount percentage. Partners have regular expression patterns that identify the complete set of URIs for the partner, or alternatively identify the partner from a URI. Partners have subscriptions by parties. Parties have IP counts and limit values. Partners also own access rules (combinations of URI patterns and access types.



## Partners

GET /partners

* returns JSON array of all partners

[   
 {"partnerId": "tair","name": "TAIR",  
 "logoUri": "<https://s3-us-west-2.amazonaws.com/pw2-logo/logo2.gif>",  
 "termOfServiceUri": "<https://www.arabidopsis.org/doc/about/tair_terms_of_use/417>"  
 },  
 {"partnerId": "yfd", "name": "YFD",   
 "logoUri": "<https://s3-us-west-2.amazonaws.com/pw2-logo/yfd.png>",  
 "termOfServiceUri": "<https://www.google.com/intl/en/policies/terms/?fg=1>"  
 }  
]

GET /partners?partnerId=<id>

* where <id> is the partner id for the partner
* returns JSON array with a single partner identified by the id

[   
 {"partnerId": "tair","name": "TAIR",  
 "logoUri": "<https://s3-us-west-2.amazonaws.com/pw2-logo/logo2.gif>",  
 "termOfServiceUri": "<https://www.arabidopsis.org/doc/about/tair_terms_of_use/417>"  
 }  
]

GET /partners?name=<value>

* where <value> is a unique partner name
* returns JSON array with a single partner identified by the name

[   
 {"partnerId": "tair","name": "TAIR",  
 "logoUri": "<https://s3-us-west-2.amazonaws.com/pw2-logo/logo2.gif>",  
 "termOfServiceUri": "<https://www.arabidopsis.org/doc/about/tair_terms_of_use/417>"  
 }  
]

GET /partners?uri=<uri>

* where <uri> is one of the partner URIs that identifies the partner
* returns JSON array with a single partner pattern that matches the input URI

[   
 "partnerId": "yfd",  
 "sourceUri": "[demoyfd.arabidopsis.org](http://demoyfd.arabidopsis.org/)",  
 "targetUri": "[http://back-demoyfd.steveatgetexp.com](http://back-demoyfd.steveatgetexp.com/)"  
 }  
]

POST /partners

* inserts a partner with form data including id, name, and the optional logoUri and termOfServiceUri fields
* returns 201 (CREATED) and the JSON array containing the object if successful, 400 (BAD\_REQUEST) and a JSON array of error messages if failed

PUT /partners

* body must include partnerId to identify the partner to update
* updates a partner with form data including name (must be unique in the set of partner names), logoUri, and termOfServiceUri fields
* returns JSON array of the updated partner or an empty array if the update failed

DELETE /partners?partnerId=<id>

* where <id> is the unique identifier for the partner
* deletes the partner identified by the id
* returns JSON message array

[{"success":"delete complete"}]

## Descriptions

The description is a section in the subscription user interface that shows benefits or other details about a particular type of subscription. The description type identifies the specific section and can be "def" (a default section), "individual" for an individual subscription, "commercial" for a commercial subscription, or "institution" for an institutional subscription. The header is the text that appears in the section header. See Description Items for the text that appears in the section itself. This feature of the API lets a partner brand the subscription pages with marketing material specific to the partner system and business model.

GET /partners/descriptions

* returns JSON array of all descriptions for all partners

[ {"partnerId": "yfd", "descriptionType": "def", "header": "Default for Test Partner"},  
 {"partnerId": "tair", "descriptionType": "individual",  
 "header": "Individual Subscription"   
 }  
]

GET /partners/descriptions?partnerId=<id>&header=<header>

* where <id> is the partner id for the partner and <header> is a string unique within the partner id; the combination identifies a single description
* returns JSON array with a single description identified by the id and header

[   
 {"partnerId": "yfd", "descriptionType": "def", "header": "Default for Test Partner"}  
]

GET /partners/descriptions?partnerId=<id>

* where <id> is the partner id for the partner; gets a set of descriptions for a single partner
* returns JSON array with descriptions for the partner

[ {"partnerId": "yfd", "descriptionType": "def", "header": "Default for Test Partner"},  
 {"partnerId": "yfd", "descriptionType": "individual",   
 "header": "Test Individual Subscription"  
 }  
]

GET /partners/descriptions?partnerId=<id>&includeText=true

* where <id> is the partner id for the partner; gets a set of descriptions and their description items for a single partner
* returns a special single object containing the descriptions and text lines:

{  
 "commercial":   
 {  
 "benefits":  
 [  
 "Used by most top agroscience companies",  
 "Subscription options for entire companies or employee uses",  
 "License terms appropriate for commercial uses"  
 ],  
 "id": "commercial",  
 "heading": "Commercial Subscription Benefit"  
 },  
 "individual":   
 {  
 "benefits":  
 [  
 "Access for a single researcher",  
 "Each lab member requires their own individual subscription",  
 " Discounts when two or more individuals subscribe together "  
 ],  
 "id": "individual",  
 "heading": "Individual Subscription Benefit"  
 }  
 }

POST /partners/descriptions

* inserts a description with form data including partner id, description type (one of "def", "individual", "institution", or "commercial"), and header
* returns 201 (CREATED) and the JSON object if successful, 400 (BAD\_REQUEST) and a JSON array of error messages if failed

PUT /partners/descriptions

* body must contain the partnerId and descriptionType to identify the description to update
* updates the description identified by the partner id and description type with form data containing a new header
* returns JSON array of the updated object or an empty array if the update failed

DELETE /partners/descriptions?partnerId=<id>&descriptionType=<type>

* where <id> is the unique identifier for the partner and <type> is the unique description type that identifies the description within the partner id
* deletes the description identified by the partner id and description type
* returns JSON message array

[{"success":"delete complete"}]

## Description Items

The description items are lines of text that come under the description header in the user interface section identified in the description (see Descriptions). This allows the partner to brand the subscription page with marketing material specific to the partner system and business model.

GET /partners/descriptionItems

* returns JSON array of all description items for all partners

[ {"partnerId": "yfd", "descriptionType": "individual",  
 "itemNo": "1", "text":"This is the individual benefit #1 for test partner."  
 },  
 {"partnerId": "yfd", "descriptionType ": "individual",  
 "itemNo": "2", "text":" This is the individual benefit #2 for test partner."  
 },  
 {"partnerId": "tair", "descriptionType ": "individual",  
 "itemNo": "1", "text":"This is the individual benefit #1 for TAIR."  
 },  
 {"partnerId": "tair", "descriptionType ": "individual",  
 "itemNo": "2", "text":" This is the individual benefit #2 for TAIR."  
 }  
]

GET /partners/descriptionItems?partnerId=<id>&descriptionType=<type>  
 &itemNo=<n>

* where <id> is the partner id for the partner, <type> is a text identifier unique within the partner id, and <n> is a number that identifies and orders the item within the type; the combination identifies a single description item
* returns JSON array with a single description item identified by the id and header

[ {"partnerId": "yfd", "descriptionType": "individual",  
 "itemNo": "1", "text":"This is the individual benefit #1 for test partner."}]

GET /partners/descriptionItems?partnerId=<id>&descriptionType=<type>

* where <id> is the partner id for the partner and <type> is a text identifier unique within the partner id; the combination identifies a set of description items
* returns JSON array with the set of description item identified by the id and description type ordered by the item number

[ {"partnerId": "yfd", "descriptionType": "individual",  
 "itemNo": "1", "text":"This is the individual benefit #1 for test partner."  
 },  
 {"partnerId": "yfd", "descriptionType ": "individual",  
 "itemNo": "2", "text":" This is the individual benefit #2 for test partner."  
 }  
]

POST /partners/descriptionItems

* inserts a description item with form data including partner id, description type, item number, and item text
* returns 201 (CREATED) and the JSON object if successful, 400 (BAD\_REQUEST) and a JSON array of error messages if failed

PUT /partners/descriptionItems

* the partnerId, descriptionType, and itemNo must appear in the body to identify the item to update
* updates the item with form data containing a new item text
* returns JSON array of the updated object or an empty array if the update failed

DELETE /partners/descriptionItems?partnerId=<id>&descriptionType=<type>  
 &itemNo=<n>

* where <id> is the partner id for the partner, <type> is a text identifier unique within the partner id, and <n> is a number that identifies and orders the item within the description type; the combination identifies a single description item
* deletes the item identified by the id, type, and item number
* returns JSON message array

[{"success":"delete complete"}]

## Patterns

Each partner system has a set of URIs that you use to access the various parts of the partner site. The partner patterns are regular expressions that match those URIs. For example, "^([wW]{3}\.)?arabidopsis\.org/?" is a pattern for TAIR (finds "www.arabidopsis.org/" or "arabidopsis.org/", and so on). These patterns also uniquely identify partners. Each source corresponds to a target pattern that the proxy server uses to rewrite the URL to access the database resource.

Please note that the source pattern is part of the key of the object and thus cannot be updated. If you want a different pattern than one already in the database, delete the existing pattern and insert the new pattern rather than updating the existing pattern.

GET /partners/patterns

* returns JSON array of all patterns for all partners

[  
 {"partnerId": "yfd",   
 "sourceUri": "[testyfd.com](http://testyfd.com/)",  
 "targetUri": "[http://back-prod.testyfd.com](http://back-prod.testyfd.com/)"  
 },  
 {"partnerId": "yfd",   
 "sourceUri": "[demoyfd.arabidopsis.org](http://demoyfd.arabidopsis.org/)",  
 "targetUri": "[http://back-demoyfd.steveatgetexp.com](http://back-demoyfd.steveatgetexp.com/)"  
 },  
 {"partnerId": "tair",   
 "sourceUri": "[proxy2.steveatgetexp.com](http://proxy2.steveatgetexp.com/)",  
 "targetUri": "[http://back-prod.arabidopsis.org](http://back-prod.arabidopsis.org/)"  
 }  
]

GET /partners/patterns?partnerId=<id>&sourceUri=<pattern>

* where <id> is the partner id for the partner and <pattern> is a unique pattern string within the partner id; the combination identifies a single pattern
* returns JSON array with a single pattern identified by the id and pattern

[  
 {"partnerId": "yfd",   
 "sourceUri": "[testyfd.com](http://testyfd.com/)",  
 "targetUri": "[http://back-prod.testyfd.com](http://back-prod.testyfd.com/)"  
 }  
]

POST /partners/patterns

* inserts a pattern with form data including partner id, source URI, and target URI
* returns 201 (CREATED) and the JSON object if successful, 400 (BAD\_REQUEST) and a JSON array of error messages if failed

PUT /partners/patterns

* the partnerId and sourceUri must appear in the body to identify the pattern to update
* updates the pattern with form data containing a new target URI
* returns JSON array of the updated object or an empty array if the update failed

DELETE /partners/patterns?partnerId=<id>&sourceUri=<pattern>

* where <id> is the unique identifier for the partner and <pattern> is the unique regular expression pattern that identifies the pattern within the partner id
* deletes the pattern identified by the id and pattern
* returns JSON message array

[{"success":"delete complete"}]

## Terms

A subscription term is a collection of attributes describing a particular subscription option. It has the period (in months, so 12 is an annual subscription, for example), the price in dollars and cents, a group-discount percentage (.1 means a 10% discount for group subscriptions, for example), and a description of the term that should appear in the user interface. A partner specifies the various subscriptions their business model offers to brand the Subscription page with the pricing.

Note that the combination of partner ID and period identifies the term, so you can't update the period; instead, delete the existing term and insert a new term with a different period.

GET /partners/terms

* returns JSON array of all terms for all partners

[  
 {"partnerId": "yfd", "period": 60, "price": "100.00",  
 "groupDiscountPercentage": "0.00", "description": "Two Month"  
 },  
 {"partnerId": "yfd", "period": 240, "price": "400.00",  
 "groupDiscountPercentage": "10.00", "description": "Eight Months"  
 },  
 {"partnerId": "yfd", "period": 1095, "price": "1800.00",   
 "groupDiscountPercentage": "20.00", "description": "Three Year"  
 },  
 {"partnerId": "tair", "period": 30, "price": "9.80",  
 "groupDiscountPercentage": "10.00", "description": "1 month"  
 },  
 {"partnerId": "tair", "period": 365, "price": "98.00",  
 "groupDiscountPercentage": "10.00", "description": "1 year"  
 },  
 {"partnerId": "tair", "period": 730, "price": "196.00",  
 "groupDiscountPercentage": "10.00", "description": "2 years"  
 }  
]

GET /partners/terms?partnerId=<id>&period=<period>

* where <id> is the partner id for the partner and <period> is a unique period within the partner id; the combination identifies a single term
* returns JSON array with a single term identified by the id and pattern

[  
 {"partnerId": "yfd", "period": 60, "price": "100.00",  
 "groupDiscountPercentage": "0.00", "description": "Two Month"  
 }  
]

POST /partners/terms

* inserts a term with form data including partner id, period, price, group discount percentage, and description
* returns 201 (CREATED) and the JSON object if successful, 400 (BAD\_REQUEST) and a JSON array of error messages if failed

PUT /partners/terms?partnerId=<id>&period=<period>

* the partnerId and period need to be in the body to specify which term to update
* updates the term with form data containing values for price, group discount percentage, and description
* returns JSON array of the updated object or an empty array if the update failed

DELETE /partners/patterns?partnerId=<id>&period=<period>

* where <id> is the unique identifier for the partner and <period> is the unique period that identifies the term, within the partner id
* deletes the term identified by the id and period
* returns JSON message array

[{"success":"delete complete"}]

# Parties

A *party* is an entity that is some kind of system participant. A party has an id and a type and a set of IP ranges, continuous ranges of IPv4 or IPv6 addresses, that constitute the set of IP addresses that the party will subscribe when they subscribe to a partner. The IP range has a unique id, a starting IP address, and an ending IP address.

* **/parties** # all parties
* **/parties?partyId=<id>** # a specific party
* **/parties?partyType=<type>** # all parties of the type
* **/parties/ipranges** # all ip ranges
* **/parties/ipranges?partyId=<id>** # all the IP ranges for a party
* **/parties/ipranges?ipRangeId=<id>** # a specific IP range

# Subscriptions

A *subscription* is an agreement by a party to take and pay for access to a partner system. A subscription has a set of transactions (initial subscription, renewal, refund). Each subscription has a set of transactions that shows the transaction history of the subscription (initial subscription, renewals, and refunds).

The subscription has a party ID, a partner ID, a start date, and end Date, and a subscription ID. The combination of party ID and partner ID uniquely identifies the subscription; the subscription ID provides an alternate, single-number identifier for the subscription.

* **/subscriptions** # all subscriptions
* **/subscriptions?partnerId=<id>** # all subscriptions for a partner
* **/subscriptions?subscriptionId=<id>** # a specific subscription
* **/subscriptions/active** # all active subscriptions
* **/subscriptions/active?partyId=<id>** # all active subscriptions for a party
* **/subscriptions/active?partnerId=<id>** # all active subscriptions for a partner
* **/subscriptions/active?ip=<ip>?partnerId=<id>** # all active subscriptions for a partner
* **/subscriptions/active?partnerId=<id>&partyId=<id>** # all active subscriptions for a partner and party with an ip address in a subscribed ip range
* **/subscriptions/transactions** # all subscription transactions
* **/subscriptions/transactions/<id>** # a specific subscription transaction

# Meters

A *meter* is a count of page views. The meter subsystem comprises a set of partner-ip-address counts and a set of limits to enforce on those counts. The user accesses the system, which increments the counts and enforces the limits.

The count has a partner ID, an IP address, and an integer count. The limit has a partner ID, a name, and an integer limit value representing the count of page views at which to limit access. <ip> is an IPv4 or IPv6 internet protocol address when it occurs in the URI.

* **/meters** # all meter counts
* **/meters?partnerId=<id>** # all meter counts for a partner
* **/meters/ip** # all ip meter counts
* **/meters/ip?partnerId=<id>** # all ip meter counts for a partner
* **/meters/ip?partnerId=<id>&ip=<ip>** # the meter count for an ip accessing a partner
* **/meters/ip/<ip>/increment?partnerId=<id>** # increment the meter count for an ip accessing a partner
* **/meters/ip/limit&partnerId=<id>>&ip=<ip>** # check the status of an ip accessing a partner, returns OK, WARNING, BLOCK
* **/meters/limits/warningLimit?partnerId=<id>** # warning limit for a partner
* **/meters/limits/meteringLimit?partnerId=<id>** # metering limit for a partner

# Payment

Payment is the process of paying for a subscription. Payment interfaces to stripe.com to allow the user to pay for a subscription. The API for payment persists no data in the database. The URI supports only the POST operation with appropriate data in the body from the subscription user interface.

* **/payments** # POST a payment

# Authentication

*Authentication* is the process of verifying the identity of a user. The Subscription Management System uses a username-and-password authentication scheme as well as a key token scheme for authenticating callers of the API.

The username uniquely identifies the party, and supplying a password that matches the stored password verifies that identity. The user-password also corresponds to an optional email address and organization name.

* **/authentications** # all the username-password combinations for parties
* **/authentications?username=<username>** # retrieve, update, delete specific username
* **/authentications/login** # determine if logged in from cookie, otherwise authenticate using password

The API key authenticates a caller of the API and authorizes the calling of any API method. Phoenix creates the API keys and distributes them to calling programs as required.

* **/apikeys** # all the API key tokens

# Authorization

*Authorization* is the process of granting or refusing access to a partner. The Subscription Management System specifies partners as sets of URIs identified by regular expression patterns.

A pattern is a regular expression pattern identified by an id. An access type is a content type identified by an id with a name. An access rule relates a partner to a pattern and an access type and consists of the three ids from those objects plus a single-number, unique identifier for the rule. The access API call determines whether to authorize access to a URI for a party by finding the appropriate access rule that covers the URI; if no such rule exists, the default is to authorize the request (free content).

* **/authorizations/patterns** # all URI patterns
* **/authorizations/patterns/<id>** # a specific URI pattern
* **/authorizations/accessTypes** # all access types
* **/authorizations/accessTypes/<id>** # a specific access type
* **/authorizations/accessRules** # all access rules
* **/authorizations/accessRules/<id>** # a specific access rule
* **/authorizations/access?ip=<ip>&url=<uri>&partyId=<id>&partnerId=<id>** # whether the party or ip address has access for a given URI for a given partner; returns OK, Warning, NeedSubscription
* **/authorizations/subscription? ip=<ip>&url=<uri>&partyId=<id>&partnerId=<id>** # whether the party or ip address has access for a given URI for a given partner; returns OK, Warning, NeedSubscription

# Logging

*Logging* is the process of persisting a description of some kind of activity starting at a specific date and time and ending at a specific date and time. The Subscription Management System currently logs page views and sessions. A page view is an access of a partner with a URI. A session is a set of possibly empty related page views. Note that a session is not partner-specific.

* **/session-logs/sessions** # all sessions
* **/session-logs/sessions?startDatetime=<date>&endDatetime=<date>  
  &partyId=<id>&ip=<ip>** # selected set of sessions based on some combination of parameters, all of which are optional
* **/session-logs/sessions/create** # create a new session
* **/session-logs/page-views** # all page views
* **/session-logs/page-views?startDatetime=<date>&endDatetime=<date>  
  &partyId=<id>&ip=<ip>** # all page views between two dates, inclusive, and/or associated with a session for a specified party, and/or associated with a session with a specified ip address
* **/session-logs/page-views/create** # create a new page view in a session, set end date of session; body contains session id, page-view date-time, ip address