RFRain REST API Reference Guide

Version 7.00.13.39

Date: 2/08/2024

Copyright 2024 RFRain

Content

OVERVIEW..... 03

SESSION KEY APIS

get_sessionkey..... 08
extend_sessionkey_lifetime..... 10
destroy_sessionkey..... 11
is_sessionkey_valid..... 12

TAG INFORMATION APIS

get_list_of_active_zones..... 14
get_list_of_tags..... 15
get_list_of_tags_status..... 16
get_list_of_tags_history_status..... 18
get_tag_status_by_tagnumb..... 20
get_gps_status_by_tagnumb..... 22
add_tag_status..... 24
get_history_by_tagnumb..... 26
get_subzones..... 28
set_tag_custom_information..... 29
get_tag_custom_information..... 31
remove_tag_custom_information..... 33
cleanup_table_data..... 34
start_monitoring_new_tags_history..... 35
start_monitoring_new_tags_jitter..... 36
get_latest_tags_history..... 37
get_latest_tags_jitter..... 39

READER CONTROL APIS

get_reader_status..... 42

ALERT CONTROL APIS

start_alert_engine..... 47
stop_alert_engine..... 48
define_alert..... 49
start_alert..... 52
stop_alert..... 53

Content

start_all_alerts..... 54
stop_all_alerts..... 55
delete_alert..... 56
list_alerts..... 57
define_alert_group..... 59
delete_alert_group..... 60
list_alert_groups..... 61
add_alert_email_entry_to_group..... 63
remove_alert_email_entry_from_group..... 64
add_alert_api_entry_to_group..... 65
remove_alert_api_entry_from_group..... 66
add_notify_group_to_alert..... 67
remove_notify_group_from_alert..... 68

SYSTEM CONTROL APIS

get_system_users..... 70
add_user..... 71
delete_user..... 72
change_user_password..... 73
change_user_level..... 74

FEATURE CONTROL APIS

camera_set_retention..... 76
camera_event_association..... 79

CLOUD SPECIFIC APIS

define_reader_name..... 82
define_reader_group..... 83
delete_reader_group..... 84
list_reader_groups..... 85
add_reader_to_group..... 86
remove_reader_from_group..... 87
list_all_readers_by_group..... 88
list_readers_in_group..... 89
find_group_reader_is_in..... 90
get_reader_status_by_group..... 91

API OVERVIEW

SessionKey Access

All REST API calls require the usage of an api 'sessionkey', with the exception of the call to acquire the API key itself. If an API is called without a valid and active API sessionkey included, the call will fail.

The API key is obtained by calling 'get_sessionkey'. If the call is successful, an active sessionkey is returned that must be used for all subsequent calls. The sessionkey that is returned is tagged on the system with a 30 minute lifetime. If the key remains idle for more than 30 minutes, the sessionkey is automatically deleted by the system. Every call to the API with a valid sessionkey causes the lifetime timer to be reset back to 0. Thus an acquired sessionkey may be used for the entirety of an active session, then will gracefully expire after 30 minutes of inactivity (unless the 'destroy_sessionkey' method is called, which is preferred to do when you are done with your key).

The 'destroy_sessionkey' method will immediately destroy the active sessionkey and terminate access using that key. If the key has expired, or the 'destroy_sessionkey' has been called on a particular sessionkey, a new sessionkey must be generated before API access is granted. It is strongly recommended that the 'destroy_sessionkey' is called at the end of program usage.

SessionKey Security

SessionKeys themselves contain security information that locks the SessionKey to your particular session. The key is specific to your computing device/application. If anyone takes an active key and attempts to use it on another device, the key will fail authentication upon the next call.

Additionaly, the key itself is encrypted and contains within it a copy of the stored reference of the key on the server. If the key is tampered with in any way, it will fail authentication upon the next call. The key also contains within it an additional shared key used to encrypt data being transmitted across an open network (if the feature is enabled). This is done prior to the transmission of any data to and from a reader or service provided, and before being encrypted into a standard HTTPS/SSL transmission. This results in a 'double' encryption for your data protection.

API Interaction

While the REST API was designed to interact with JavaScript calls, it can be used by any program that can make and interpret a REST API return result set. The returned data format is in industry standard JSON formatted output. All data submitted to the REST API must also be in industry standard JSON format. HTML headers will also be set to indicate success or failure.

API Response Contents

All REST API call responses contain the following:

- 'request' parameter
  This contains the copy of the API that was called
- 'success' parameter
  This contains a boolean value, TRUE or FALSE, that signifies the success or failure of the API call.
- 'message' parameter
  This contains a response message detailing the results of the API call. If there was an error, the error message will be contained within this parameter.
- 'results' parameter
  This contains the results of the API call (if successful). The contents will vary based upon the call made. If the call was not successful, then this field will be empty.

API Call Failures

The RFRain API uses the industry standard HTTP header codes to indicate success or failure of the API call. In particular, codes 400, 401, 404, and 200

HTTP Code 400:
Per industry standards, if an API call is made missing the required sessionkey parameter, a code 400 (Bad Request) is sent back.

HTTP Code 401:
If a sessionkey is provided that fails validation, then the error code 401 (Unauthorized) is returned to the client.

HTTP Code 404:
If an API call is made to an API routine that does not exist, an error code of 404 (Not Found) is returned to the client. Note: there is some debate in the industry at this time on whether to use this code, or 503 (Service Unavailable), but for now, RFRain will use the 404 code until the industry standard recommendation changes.

HTTP Code 200:
If the API client call was processed by the API Manager properly (regardless of returned content) then this code is sent.

Usage of these codes is an industry standard practice that allows for integration of third party applications and authentication functions while maintaining compatibility.

Making an API Call

Interaction with the RFRain API Interface is done exclusively with JSON formatted system calls. These calls may be originated from Java, Javascript, ASP.NET, or any other system that can interact with REST API's using the industry standard JSON format. For explanation in this document, all examples are given in Javascript format, using the JQuery framework, but modifying it to match your interface of choice should be very straight forward.

Your First Call: Acquiring your SessionKey

Your first call to the API should be to obtain your API SessionKey. This key will be used in all interactions going forward. In this API call, a simple base64 encoding is used help obscure your login credentials using the built-in JavaScript 'btoa' (Base64 Encoding) feature before sending them across the network inside an SSL encrypted connection to the server. This adds another layer of protection.

Example (JavaScript, JQuery style AJAX call):

```
var myEmail = btoa('somebody@somewhere.com');
var myPasswd = btoa('yOur3xcell@ntP@sswOrd!');
var myCompany = 'mycompany';
var myData = { 'email': myEmail, 'password': myPasswd, 'cname': myCompany };

$.ajax({
    url: 'https://(remote_site)/rfrainapi.php/get_sessionkey',
    dataType: 'json',
    type: 'POST',
    data: myData,
    success: function(r) {
        if(r.success) { myKey = r.results.sessionkey; }
    }
});
```

Here in this example, we create an object called 'myData' that contains three key/value pairs, 'email', 'password', and 'cname' which are filled with the encrypted data and your company name. This data object is passed to the API using the POST method on an HTTPS secure connection (using the JQuery framework method). If successful, you will be passed back from the API an object containing key/value pairs with the results of your query.

The returned object has the format:

```
request:   'get_sessionkey'
success:   true/false
message: (api call detailed result or error message)
results:    { 'sessionkey': (api_assigned_sessionkey) }
```

In this case, if your call was completed successfully, the object returned will contain the output similar to the above, and the 'results' object will contain a key-value pair containing your active sessionkey. All the API interactions work in this manner.

SSL Certificates

All API interactions must use the 'HTTPS' secure connection. However, each reader has it's own selfsigned SSL certificate. Your software must be able to send and receive from an HTTPS secure connection while using self-signed certificates. If desired, custom certificates can be installed on the readers. If you desire this, talk with your sales representative or call RFRain Support.

API Command Availablility

Different Reader Models support a varying number of the API calls. All the calls, when available, and their basic parameters are the same across all reader platforms, however not all calls are available on all readers. Some readers do not support some features, while others support more advanced features that may include additional options on the API call. The best place to find the latest API specifications for your current reader model and system version, is by looking in the APITester Application's online help. It will contain the API specification available in that model/os release, how to use them, and what to expect as a response from the API processor.

Some features are only available in the Enterprise version, using the Gateways, or in the Cloud Service. There are also many features that are only available in the SmartReader© series of readers. Be sure to watch the HTTP Headers when you make a call to an uknown reader model in order to handle error responses for calls to unsupported API's. One consistent API call that is available across all the reader models (as of ZoneManager, v5.20) is the 'get_reader_status' API call. This call among other things, will return to you the reader make, model, OS, and ZoneManager version information.

Some features may be included in the base system on the reader, but are only enabled via a license key that is purchased with your reader. This license key can be upgraded at any time to add access to more features. If an API call is made to a feature that is not enabled (does not have the proper license) the API call will succeed (HTTP code 200) but will show failure in the JSON return set (r.success = false), and the message parameter will explain that an additional license is needed.

Session Key APIs

get_sessionkey

The first call should be to obtain an active API sessionkey. This is done using the get_sessionkey call.

Synopsis:
get_sessionkey

Input:
email: System Login - base64 encoding of your assigned login email credential
cname: your company name as assigned by your license
password: base64 encoding of your assigned login password

Output:
request: 'get_sessionkey'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:
{
sessionkey: (your encrypted sessionkey)
userlevel: (your level of system access)
}

Notes:

- The CName is the company name as assigned in your license.
- Assigned login credentials are created through the ZoneManager interface, or through the System API create user API call.
- Data submitted to this method (and only this method) should be base64 encoded to prevent clear-text information from transversing the network. In JavaScript, there is a built-in function to handle this, it is 'btoa' to encrypt.
- Data returned from this method is NOT base64 encoded. Only the data submitted for these two fields should be encrypted.

- Embedded in the SessionKey is encrypted data that is specific to your connection, authentication parameters, and lifetime. If anyone attempts to use this key other than your specific device, or tries to use the key at a later time and the connection parameters do NOT match the encrypted information, or the key information does not match the stored connection information, then the key is marked invalid and discarded, causing a request for a new key to be done before continued access. In this fashion, if anyone attempts to use a key 'grabbed' by observing network traffic, the key will not be accepted, and the existing key will be destroyed. This helps to prevent 'man-in-the-middle' network hack attempts.
- OAUTH and OAUTH2 are not available with this version of the REST API, but are under development and should be available in a subsequent release, as well as AES256 bit encryption for JSON data transfer, and hash values for returned data to ensure integrity (caller selectable).

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       email: btoa('somebody@somewhere.com'),
       password: btoa('yOour3xcell@ntP@sswOrd!'),
       cname: 'MyCompany'
       };

$.ajax({
       url:          'https://(remote_site)/rfrainapi.php/get_sessionkey',
       dataType:     'json',
       type:         'POST',
       data:         myData,
       success:      function(r) {
              if(r.success) {
                     myKey = r.results.sessionkey;
                     myLevel = r.results.userlevel;
              };
       }
});
```

extend_sessionkey_lifetime

There are times when an application is stuck waiting for processing to be done, or user interaction. If the 30 minute sessionkey lifetime threshold is reached, the sessionkey could be expired while your application is still processing information. To extend the lifetime (i.e. start the lifetime counter over again at 0), use this API call.

Synopsis:

extend_sessionkey_lifetime
Input:
sessionkey: (an active Sessionkey)

Output:
request: 'extend_sessionkey_lifetime'
success: TRUE/FALSE
message: (api call detailed result or error message)
results: { }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not

---

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/extend_sessionkey_lifetime',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { didWork = r.success; }
});
```

destroy_sessionkey

No further calls can be made with this sessionkey after this call is made, as the sessionkey will be removed from being considered valid. It is strongly suggested that this API method be called at the end of application usage for security reasons.

Synopsis:

destroy_sessionkey
Input:

sessionkey:	(an active Sessionkey)

Output:

request:	'destroy_sessionkey'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not.

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/destroy_sessionkey’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { didWork = r.success; }
});
```

is_sessionkey_valid

Sometimes you may wish to check to see if your key is still valid. This call will verify the key's integrity and return true/false if the key is still a good key. This call will (upon success) also extend the lifetime of the key by resetting the counter to 0.

Synopsis:

is_sessionkey_valid

Input:
sessionkey: (an active Sessionkey)

Output:
request: 'is_sessionkey_valid'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:
{
is_valid: TRUE/FALSE
}

Notes:

- The 'is_valid' result returned from this call is of type 'boolean' (i.e. true/false)

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
	url: 'https://(remote_site)/rfrainapi.php/is_sessionkey_valid',
	dataType: 'json',
	type: 'POST',
	data: myData,
	success: function(r) {
		if(r.success) { myKeyValid = r.results.is_valid; };
	}
});
```

Tag Information APIs

get_list_of_active_zones

This call will return an object that contains multiple zone_name/zone_type pairs of the active, defined zones and their types as configured through the ZoneManager interface.

Synopsis:

get_list_of_active_zones
Input:
sessionkey: (an active Sessionkey)

Output:
request: 'get_list_of_active_zones'
success: TRUE/FALSE
message: (api call detailed result or error message)
[groupname]: (optional: groupname, used without the '[' and ']' characters)
[readername]: (optional: readername, used without the '[' and ']' characters)
results:
{
{
zone_name: (string: zone name)
zone_type: (string: zone type)
}
...
}

Notes:

- The zone types can be 'reader', 'database', or 'system'
- The optional groupname and readername parameters are available within the Cloud Service only, and allow you to narrow your query to a given reader, or group of readers

---

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
	url: 'https://(remote_site)/rfrainapi.php/get_list_of_active_zones',
	dataType: 'json',
	type: 'POST',
	data: myData,
	success: function(r) {
		if(r.success) {
			$.each(r.results, function(k,v) {
				myZoneName = v.zone_name;
				myZoneType = v.zone_type;
			});
		};
	}
});
```

get_list_of_tags

This call will return an object that contains of all tags active in the system; one record per tag. Warning: This can return a large amount of data, so caution should be used with this API call.

Synopsis:

get_list_of_tags
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
[groupname]:	(optional: groupname, used without the '[' and ']' characters)
[readername]:	(optional: readername, used without the '[' and ']' characters)

Output:

request:	'get_list_of_tags'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	

```
{
 {
```

tagnumb:	(the tag number)
tagname:	(the tag name)

```
 }
...
}
```

Notes:

- The Database Name by default is 'RfrainDB', but if you have created additional databases, then it would be the name of the database you wish to return the data from.
- The optional groupname and readername parameters are available within the Cloud Service only, and allow you to narrow your query to a given reader, or group of readers.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
        sessionkey: myKey,
        dbname: myDBName
        };

$.ajax({
        url: 'https://(remote_site)/rfrainapi.php/get_list_of_tags',
        dataType: 'json',
        type: 'POST',
        data: myData,
        success: function(r) {
                if(r.success) {
                        $.each(r.results, function(k,v) {
                                myTagNumb = v.tagnumb;
                                myTagName = v.tagname;
                        });
                };
        }
});
```

get_list_of_tags_status

This call will return an object that contains of all tags active in the system; one record per tag.
Warning: This can return a large amount of data, so caution should be used with this API call.

Synopsis:

get_list_of_tags_status
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
readername:	(readername, optional)
groupname:	(groupname, optional)
altsubzone:	(subzone, optional)

Output:

category:	tag_api
request:	'get_list_of_tags_status'
success:	TRUE/FALSE
message:	(api call detailed result or error message)

results:

```
{
  {
```

sr.no:	(the record number)
tagnumb:	(the tag number)
tagname:	(the tag name)
subzone:	(the subzone or antenna name)
detectstat:	(the status PRES/MISS)
SS:	(the signal strength when tag is present)
lat:	(the latitude information when tag is present)
lng:	(the longitude information when tag is present)
access:	(access time when tag is present)
access_utc:	(access time in UTC when tag is present)
readerid:	(readerid when tag is present)
readername:	(readername when tag is present)
groupname:	(groupname when tag is present)
current_detectstat:	(current status of tag)
current_subzone:	(the current subzone or antenna name)
current_SS:	(the current signal strength)
current_lat:	(the latitude information)
current_lng:	(the longitude information)
current_access:	(current access time of tag)
current_access_UTC:	(current access time in UTC of tag)
current_readerid:	(current readerid of the tag)
current_readername:	(current readername of the tag)
current_groupname:	(current groupname of the tag)

```
 }
...
}
```

Notes:

- The Database Name by default is 'RfrainDB', but if you have created additional databases, then it would be the name of the database you wish to return the data from.
- The optional groupname and readername parameters are available within the Cloud Service only, and allow you to narrow your query to a given reader, or group of readers or by subzone.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName
       readername: myReaderName
       groupname: myGroupName
       altsubzone: mySubzoneName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/get_list_of_tags_status',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                     mySrNo = v.sr.no;
                            myTagNumb = v.tagnumb;
                            myTagName = v.tagname;
                            subzone = v.subzone;
                     detectstat = v.detectstat;
                            SS = v.SS;
                            lat = v.lat;
                            lng = v.lng;
                            access = v.access;
                            access_utc = v.access_utc;
                            readerid = v.reader;
                            readername = v.readername;
                            groupname = v.groupname;
                            current_detectstat = v.current_detectstat;
                     current_subzone = v.current_subzone;
                     current_SS = v.current_ss;
                            current_lat = v.current_lat;
                            current_lng = v.current_lng;
                            current_access = v.current_access;
                            current_access_utc = v.current_access_utc;
                            current_readerid = v.current_reader;
                            current_readername = v.current_readername;
                            current_groupname = v.current_groupname;
                     });
              };
       }
});
```

get_list_of_tags_history_status

This call will return an object that contains of all tags active in the system; one record per tag. Warning: This can return a large amount of data, so caution should be used with this API call.

Synopsis:

get_list_of_tags_history_status
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
groupname:	(groupname, optional)
readername:	(readername, optional)
altsubzone:	(subzone, optional)
startaccess:	(start access time in UTC, optional)
endaccess:	(end access time in UTC, optional)
maxrecords:	(max history records to fetch, optional, default=5000)

Output:

category:	tag_api
request:	'get_list_of_tags_history_status'
success:	TRUE/FALSE
message:	(api call detailed result or error message)

results:
{
max_access_time_utc: maximum (latest) access time in UTC from all avaialble records
max_history_records: Total (maximum) records found
history_results (start): history results subset {

sr.no:	(record count number)
tagnumb:	(the tag number)
tagname:	(the tag name)
lat:	(the latitude when status is PRES)
lng:	(the longitude when status is PRES)
detectstat:	(the status)
subzone:	(the subzone or antenna name)
SS:	(the signal strenght when tag is present)
access:	(access time when tag is present)
access_utc:	(access time in UTC when tag is present)
readerid:	(readerid when tag is present)
readername:	(readername when tag is present)
groupname:	(groupname when tag is present)
current_detectstat:	(current status of tag)
current_subzone:	(the current subzone or antenna name)
current_SS:	(the current signal strenght when tag is present)
current_lat:	(the current latitude)
current_lng:	(the current longitude)
current_access:	(current access time of tag)
current_access_utc:	(current access time in UTC of tag)
current_readerid:	(current readerid of the tag)

current_readername: (current readername of the tag)
current_groupname: (current groupname of the tag)
history_results (end): } end of history results subset
}

Notes:

- The Database Name by default is 'RfrainDB'.
- The optional groupname, readername and subzone parameters are available within the Cloud Service only, and allow you to narrow your query to a given reader, or group of readers or by subzone. Output is displayed in descending order of access_utc time. Maximum history records found and latest access time in UTC have been returned.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
        sessionkey: myKey,
        dbname: myDBName
        readername: myReaderName
        groupname: myGroupName
        altsubzone: mySubzoneName
        startaccess: myStartAccessTimeUTC
        endaccess: myEndAccessTimeUTC
        maxrecords: myMaxHistoryRecords
                };
$.ajax({
                url:‘https://(remote_site)/rfrainapi.php/get_list_of_tags_history_status’,
                dataType: 'json',
                type: 'POST',
                data: myData,
                success: function(r) {
                        if(r.success) {
                                $.each(r.results, function(k,v) {
                                        myMaxAccessTimeUTC = v.max_access_time_utc;
                                        myMaxHistoryRecords = v.max_history_records;
                                        $.each(r.results.history_results, function(k,v) {
                                                mySrNo = v.sr.no;
                                                myTagNumb = v.tagnumb;
                                                myTagName = v.tagname;
                                                detectstat = v.detectstat;
                                                subzone = v.subzone;
                                                SS = v.SS;
                                                lat = v.lat;
                                                lng = v.lng;
                                                access = v.access;
                                                access_utc = v.access_utc;
                                                readerid = v.reader;
                                                readername = v.readername;
                                                groupname = v.groupname;
                                                current_detectstat = v.current_detectstat;
                                                current_subzone = v.current_subzone;
                                                current_SS = v.current_ss;
                                                current_lat = v.current_lat;
                                                current_lng = v.current_lng;
                                                current_access = v.current_access;
                                                current_access_utc = v.current_access_utc;
                                                current_readerid = v.current_reader;
                                                current_readername = v.current_readername;
                                                current_groupname = v.current_groupname;
                                        });
                                });
                                 };
                }
});
```

get_tag_status_by_tagnumb

This call will return the last recorded status and current status of a given tag.

Synopsis:

get_tag_status_by_tagnumb
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
tagnumb:	(tag number)
[readername]:	(optional: readername, used without the '[' and ']' characters)
[groupname]:	(optional: groupname, used without the '[' and ']' characters)
[altsubzone]:	(optional: subzonename, used without the '[' and ']' characters)

Output:

category:	tag_api
request:	'get_tag_status_by_tagnumb'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	
{	
tagnumb:	(the tag number)
tagname:	(the tag name)
lat:	(the latitude information when tag is present)
lng:	(the longitude information when tag is present)
detectstat:	(the status PRES/MISS)
subzone:	(the subzone or antenna name)
SS:	(the signal strength when tag is present)
access:	(access time when tag is present)
access_utc:	(access time in UTC when tag is present)
reader:	(readerid when tag is present)
readername:	(readername when tag is present)
groupname:	(groupname when tag is present)
current_lat:	(the latitude information)
current_lng:	(the longitude information)
current_detectstat:	(current status of tag)
current_subzone:	(the current subzone or antenna name)
current_SS:	(the current signal strength)
current_access:	(current access time of tag)
current_access_UTC:	(current access time in UTC of tag)
current_reader:	(current readerid of the tag)
current_readername:	(current readername of the tag)
current_groupname:	(current groupname of the tag)
}	

Notes:

- The access time is the latest time this tag was read by the reader.
- The time and date format is in the format of yyyy-mm-dd hh:mm:ss, and in the timezone as set in the ZoneManager.
- The hours in the time portion are in 24 hour notation format
- The optional groupname, readername and subzone parameters are available within the Cloud Service only, and allow you to narrow your query to a given reader, or group of readers or by subzone.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       tagnumb: myTagNumber
       readername: myReaderName
       groupname: myGroupName
       altsubzone: mySubzone
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/get_tag_status_by_tagnumb',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {,
              if(r.success) {
                           myTagNumb = v.tagnumb;
                           myTagName = v.tagname;
                           lat = v.lat;
                           lng = v.lng;
                           detectstat = v.detectstat;
                           subzone = v.subzone;
                           SS = v.SS;
                           access = v.access;
                           access_utc = v.access_utc;
                           readerid = v.reader;
                           readername = v.readername;
                           groupname = v.groupname;
                           current_lat = v.current_lat;
                           current_lng = v.current_lng;
                           current_detectstat = v.current_detectstat;
                           current_subzone = v.current_subzone;
                           current_SS = v.current_ss;
                           current_access = v.current_access;
                           current_access_utc = v.current_access_utc;
                           current_readerid = v.current_reader;
                           current_readername = v.current_readername;
                           current_groupname = v.current_groupname;
              };
       }
});
```

get_gps_status_by_tagnumb

This call will return the last recorded GPS status of a given tag.

Synopsis:

get_gps_status_by_tagnumb
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
tagnumb:	(tag number)
[readername]:	(optional: readername, used without the '[' and ']' characters)
[groupname]:	(optional: groupname, used without the '[' and ']' characters)

Output:

category:	tag_api
request:	'get_gps_status_by_tagnumb'
success:	TRUE/FALSE
message:	(api call detailed result or error message)

results:
{

tagnumb:	(the tag number)
tagname:	(the tag name)
lat:	(the latitude information when tag is present)
lng:	(the longitude information when tag is present)
subzone:	(the subzone or antenna name)
detectstat:	(the status PRES/MISS)
SS:	(the signal strength when tag is present)
access:	(access time when tag is present)
access_utc:	(access time in UTC when tag is present)
reader:	(readerid when tag is present)
readername:	(readername when tag is present)
groupname:	(groupname when tag is present)

}

Notes:

- The access time is the latest time this tag was read by the reader.
- The time and date format is in the format of yyyy-mm-dd hh:mm:ss, and in the timezone as set in the ZoneManager.
- The hours in the time portion are in 24 hour notation format
- The optional groupname, readername parameters are available within the Cloud Service only, and allow you to narrow your query to a given reader, or group of readers.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       tagnumb: myTagNumber
       readername: myReaderName
       groupname: myGroupName
       };

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/get_gps_status_by_tagnumb’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {,
              if(r.success) {
                            myTagNumb = v.tagnumb;
                            myTagName = v.tagname;
                            lat = v.lat;
                            lng = v.lng;
                            subzone = v.subzone;
                            detectstat = v.detectstat;
                            SS = v.SS;
                            access = v.access;
                            access_utc = v.access_utc;
                            readerid = v.reader;
                            readername = v.readername;
                            groupname = v.groupname;
              };
       }
});
```

add_tag_status

This call will save and/or update tag information for the the tags submitted as an JSON encoded array.

Synopsis:

add_tag_status
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
taglist:	(JSON Encoded array of tag objects)

Output:

category:	tag_api
request:	'add_tag_status'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- The taglist variable is a JSON encoded array of tag objects. Each tag object contains:

tagnumb	The Tag Number
tagname	The Tag Name
subzone	The Tag SubZone Name
gps	The GPS Data in Lat,Lng decimal coded notation
access	The UTC TimeStamp of This Change
ss	The Singal Strength of this reading (RSSI)
readerid	The Unique ID of the reader making this change
data	(optional) If included, this field holds any extended tag data, such as the TID

The entire multi-dimensional array is then JSON encoded and transmitted.
Ex:

```
[
{
"tagnumb": "01-02-03-04-05-06-07-08-09-0A-0B-0C",
"subzone": "ProdLine2",
"gps": "27.3366042,-82.5288915",
"access": "1566250622",
"ss": "97",
"readerid": "B827EBB52EC3"
"data": "12013BFC0006576AAC"
}
]
```

The final encoded string for this tag would look like:

```
[{“tagnumb”:”01-02-03-04-05-06-07-08-09-0A-0B-0C”,”subzone”:”ProdLine2”,
“gps”:”27.3366042,-82.5288915”,”access”:”1566250622”,”ss”:”97”,
“readerid”:”B827EBB52EC3”,”data”:”12013BFC0006576AAC”}]
```

- No results are returned from this call. The ‘success’ parameter will indicate whether or not the call was successfull.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       taglist: myTagList
       };

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/add_tag_status’,
       dataType: ‘json’,
       type: ‘POST’,
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myTotalRecords = v.totalrecords;
                            mySkippedRecords = v.skipped_records;
                            $.each(r.results.record_details, function(k,v) {
                                   myTagNumb = v.tagnumb;
                                   mySuccess = v.success;
                                   myMessage = v.msg;
                                   });
                            });
                     };
              }
       });
```

get_history_by_tagnumb

This call will return the history of a single tagnumb as an array of object value pairs, giving you tagnumb, tagname, status, subzone, SS, access time, access time in UTC, latitude, longitude, readerid, readername and groupname information. This could return a large amount of data depending on how many times the tag has been read. The data is returned in access date/time descending order with the latest information first. If groupname, readername and/or subzone is given, data will be filtered accordingly.

Synopsis:

get_history_by_tagnumb
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
tagnumb:	(tag number)
[groupname]:	(optional: groupname, used without the '[' and ']' characters)
[readername]:	(optional: readername, used without the '[' and ']' characters)
[altsubzone]:	(optional: subzone name, used without the '[' and ']' characters)
[alarmtype]:	(optional: alarmtype, used without the '[' and ']' characters)

Output:

category:	tag_api
request:	'get_history_by_tagnumb'
success:	TRUE/FALSE
message:	(api call detailed result or error message)

results:
{
 {

tagnumb:	(the tag number)
tagname:	(the tag name)
detectstat:	(the status of this tag {i.e. PRES/MISS} and the time denoted by access)
subzone:	(the subzone {antenna} from which this tag was read at the time denoted by access)
SS:	Signal strength
access:	(the date/time of the change to this state)
access_utc:	(the date/time of the change to this state in UTC)
gps_access_utc:	(the date/time of the change to this state in UTC from gps record)
lat:	(the latitude used at time of reading the tag)
lng:	(the longitude used at time of reading the tag)
readerid:	(the readerid used to read the tag)
readername:	(the readername used to read the tag)
groupname:	(the groupname used to read the tag)
alarmtype:	(the alarmtype used - mon or subz)

 }
...
}

Notes:

- The optional groupname,readername and subzonename parameters are available within the Cloud Service only, and allow you to narrow your query to a given reader, or group of readers, or bt subzone.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       tagnumb: myTagNumber
       readername: myReaderName
       groupname: myGroupName
       altsubzone: mySubzone
       alarmtype: myAlarmtype
       };

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/get_history_by_tagnumb’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myTagNumb = r.results.tagnumb;
                            myTagName = r.results.tagname;
                            myTagStatus = r.results.detectstat;
                            myTagSubZone = r.results.subzone;
                            myTagSS = r.results.ss;
                            myTagLastSeen = r.results.access;
                            myTagLastSeenUTC = r.results.access_utc;
                            myTagGpsLastSeenUTC = r.results.gps_access_utc;
                            myTagLat = r.results.lng;
                            myTagLng = r.results.lng;
                            myTagReaderId = r.results.reader;
                            myTagReaderName = r.results.readername;
                            myTagGroupName = r.results.groupname;
                            myTagAlarmType = r.results.alarmtype;
                     };
              };
       }
});
```

get_subzones

This call will return an object that contains multiple subzone names as configured through the ZoneManager interface.

Synopsis:

get_subzones
Input:

sessionkey:	(an active Sessionkey)
dbname:	(optional) database name, default is 'RfrainDB'

Output:

category:	tag_api
request:	'get_subzones'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- One or more values of defined subzones will be displayed

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey, dbname: myDbname };

$.ajax({
	url: 'https://(remote_site)/rfrainapi.php/get_subzones',
	dataType: 'json',
	type: 'POST',
	data: myData,
	success: function(r) {
		if(r.success) {
			$.each(r.results, function(k,v) {
				mySubzone = v;
			});
		};
	}
});
```

set_tag_custom_information

This call will set the values of the Tag Name, and custom columns.

There are five custom columns that may be used for holding any information you wish. These columns are typically used in conjunction with the alert process to monitor tags for specific conditions, but may be used to simply provide more information about the tag being scanned.

On the SmartReader series, these columns are named custom1, custom2, custom3, custom4, custom5 if you are using the default database configuration. You may change the names of these columns if you create a new default database for use on the SmartReader through the ZoneManager Web Interface. However, if you choose to do so, you will have to be careful to consistantly use the new database and column names throughout all the services.

Please refer to the ZoneManager documentation for more details.

Synopsis:

set_tag_custom_information

Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
tagnumb:	(tag number)
readerid:	(Reader Identity)
taginfo:	(comma-delimit list of options)

Output:

request:	'set_tag_custom_information'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- TagInfo is a comma-delimited list of "tagname", "custom1", "custom2", "custom3", "custom4", "custom5".
- No results are returned from this call, the 'success' field will tell you if it was successful or not.
- All five custom columns must be preset in the API call even if you have no value to put in them. If you have no value to put in them, send an empty string by using two single quote marks.

Example (JavaScript, JQuery style AJAX call):

```
var myTagInfo = “MyTag”,”2019-10-02”,””,””,””,””’;
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       tagnumb: myTagNumber,
       readerid: myReaderID
       taginfo: myTagInfo
       };

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/set_tag_custom_information’,
       dataType: ‘json’,
       type: ‘POST’,
       data: myData,
       success: function(r) { didWork = r.success; };
});
```

get_tag_custom_information

This call will get the values of the Tag Name, and custom columns. There are five custom columns that are used for holding any information. Please refer to the ZoneManager documentation for more details.

Synopsis:

get_tag_custom_information

Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
tagnumb:	(tag number)
readerid:	(Reader Identity)

Output:

request:	'get_tag_custom_information'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	tagnumb,tagname,custom1,custom2,custom3,custom4,custom5
results:	
{	
tagnumb:	(the tag number)
tagname:	(the tag name)
readerid:	(reader identity)
custom1:	(the custom1 information)
custom2:	(the custom2 information)
custom3:	(the custom3 information)
custom4:	(the custom4 information)
custom5:	(the custom5 information)
accessutc:	(access time in UTC (optional))
}	

Notes:

- Response to the call is "tagnumb", "tagname", "custom1", "custom2", "custom3", "custom4", "custom5", "access", "access_utc"

Example (JavaScript, JQuery style AJAX call):

```
var myTagInfo = '"E2-00-90-33-11-06-00-70-17-80-5A-07","Myag","2019-10-02","","","","","";
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       readerid: myReaderID,
       tagnumb: myTagNumber,
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/get_tag_custom_information',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myTagNumb = v.tagnumb;
                            myTagName = v.tagname;
                            myCustom1 = v.custom1;
                            myCustom2 = v.custom2;
                            myCustom3 = v.custom3;
                            myCustom4 = v.custom4;
                            myCustom5 = v.custom5;
                            myAccess = v.access;
                            myAccessUTC = v.access_utc;
                     };
              };
       }
});
```

remove_tag_custom_information

This call will remove tagnumber from the reader_default_tag table along with all custom entries. Please refer to the ZoneManager documentation for more details.

Synopsis:

remove_tag_custom_information
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')

Output:

request:	'remove_tag_custom_information'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       tagnumb: myTagNumber,
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/remove_tag_custom_information',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { didWork = r.success; };
});
```

cleanup_table_data

This call will delete all records from the table.

Synopsis:

cleanup_table_data
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')
tablename:	(tablename from which data is to be cleaned up)

Output:

request:	'cleanup_table_data'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not.
- At present, valid tablenames are reader_default and reader_default_l.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       tablename: myTablename,
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/cleanup_table_data',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { didWork = r.success; };
});
```

start_monitoring_new_tags_history

This call will clear the internal flags on the history table that control whether a tag is considered new or not. This is used in conjunction with the API call get_latest_tags_history. This call clears the flag for all existing entries in the database. Any tags that are read AFTER the flag is cleared are considered new the first time they are read.

Synopsis:

start_monitoring_new_tags_history
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')

Output:

request:	'start_monitoring_new_tags_history'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- The Database Name by default is 'RfrainDB' on the SmartReader series.
- This flag is primarily used when teamed with a PLC or polling computer to get a quick list of new tags since the last time polled.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/start_monitoring_new_tags_history',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              var FlagsCleared = r.success;
       }
});
```

start_monitoring_new_tags_jitter

This call will clear the internal flags on the jitter table that control whether a tag is considered new or not. This is used in conjunction with the API call get_latest_tags_history. This call clears the flag for all existing entries in the database. Any tags that are read AFTER the flag is cleared are considered new the first time they are read.

Synopsis:

start_monitoring_new_tags_jitter
Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, default is 'RFRainDB')

Output:

request:	'start_monitoring_new_tags_jitter'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- The Database Name by default is 'RfrainDB' on the SmartReader series.
- This flag is primarily used when teamed with a PLC or polling computer to get a quick list of new tags since the last time polled.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/start_monitoring_new_tags_jitter',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              var FlagsCleared = r.success;
       }
});
```

get_latest_tags_history

This call will a list of tags the history table if the tag is considered new. One returned in this call, then flag is cleared. This API is intended to be used in conjuction with the "start_monitoring_new_tags_history" API to enable a quick pulling method to obtains newly read tags instead of using push notifications.

Synopsis:

get_latest_tags_history
Input:
sessionkey: (an active Sessionkey)
dbname: (database name, default is 'RFRainDB')

Output:
request: 'get_latest_tags_history'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:

```
{
 {
tagnumb: (The tag number)
tagname: (The tag name)
detectstat: (The tag status (PRES/MISS))
subzone: (The tag subzone)
SS: (The tag Signal Strength)
access: (The UTC time when tag was read)
reader: (The reader from which the tag was read)
data: (The extended tag data {if any})
 }
...
}
```

Notes:

- The Database Name by default is 'RfrainDB' on the SmartReader series.
- This flag is primarily used when teamed with a PLC or polling computer to get a quick list of new tags since the last time polled.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/get_latest_tags_history',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.worked) {
                     var myTagNumb = r.results.tagnumb;
                     var myTagName = r.results.tagname;
                     var myDetectStat = r.results.detectstat;
                     var mySubzone = r.results.subzone;
                     var mySS = r.results.SS;
                     var myAccess = r.results.access;
                     var myReader = r.results.reader;
              }
       }
});
```

get_latest_tags_jitter

This feature is only relevant when readers are feeding live into the cloud. This call will a list of tags the jitter table if the tag is considered new. One returned in this call, then flag is cleared. This API is intended to be used in conjuction with the "start_monitoring_new_tags_ jitter" API to enable a quick pulling method to obtains newly read tags instead of using push notifications.

Synopsis:

get_latest_tags_jitter
Input:
sessionkey: (an active Sessionkey)
dbname: (database name, default is 'RFRainDB')

Output:
request: 'get_latest_tags_jitter'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:

{
{
tagnumb: (The tag number)
tagname: (The tag name)
detectstat: (The tag status (PRES/MISS))
subzone: (The tag subzone)
SS: (The tag Signal Strength)
access: (The UTC time when tag was read)
reader: (The reader from which the tag was read)
data: (The extended tag data {if any})
}
...
}

Notes:

- The Database Name by default is 'RfrainDB' on the SmartReader series.
- This flag is primarily used when teamed with a PLC or polling computer to get a quick list of new tags since the last time polled.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/get_latest_tags_jitter',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.worked) {
                     var myTagNumb = r.results.tagnumb;
                     var myTagName = r.results.tagname;
                     var myDetectStat = r.results.detectstat;
                     var mySubzone = r.results.subzone;
                     var mySS = r.results.SS;
                     var myAccess = r.results.access;
                     var myReader = r.results.reader;
              }
       }
});
```

Reader Control APIs

get_reader_status

This call will return the current status of the reader. This api call is also used by the Gateway, Enterprise, and Cloud services to record the 'heartbeat' of the reader. This will return whether or not the reader is scanning, the mode it is scanning in, the model, serial number, subzone names, and much more.

Synopsis:
get_reader_status

Input:
sessionkey: (an active Sessionkey)
readerid: (reader name, default is 'RfrainReader')

Output:
request: 'get_reader_status'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:
{
{ system: (the reader system subset)
reader_id: (the unique-id of the reader {typically the serial number})
reader_model: (the reader model)
name: (the name of the reader)
groupname: (the groupname of the reader)
model: (the model of the reader)
memory: (the available memory in the reader)
storage: (the available storage in the reader)
reader_op_mode: (the reader system operational mode)
reader_manage_by: (the management server)
reader_alerts_enabled: (if reader-based alerts are enabled)
reader_dbsync_enabled: (if the reader is paired and sending data to a Gateway, Enterprise, or Cloud service)
remoteaddr: (the Remote IPAddress of the reader {cloud only})
} (end system):
{ reader: (the reader subset)
reader_ip: (the IPAddress of the reader, for the SmartReader series, this will be 127.0.0.1)
reader_mode: (the current reader scanning mode)
is_alive: (if the reader is ready for scanning)
is_running: (if the reader is currently scanning)
ant_num: (the number of antennas configured)
ant_power: (the running power setting for the antennas)
ant_region: (the operational region setting)
location: (the location of the reader)
zone: (the zone name of this reader)

subzones: (a list of the antenna subzone names in JSON format as: ant#:subzonename,ant#:subzonename...)
features: (the installed features of the reader)
last_update: (the last update from the reader (cloud only))
last_update_utc: (the last update int UTC from the reader (cloud only))
} (end reader):
{ software: (the software subset)
{ software.core: (the OS software subset)
os: (the os firmware release of the reader)
zonemanager: (the ZoneManager software version installed on the reader)
backend: (the ZoneManager Reader Firmware software version installed on the reader)
} (end software.core):
{ software.engine: (the core software subset)
licensing: (The License processing software library version)
api: (The API library release version)
database: (The Database processing software library version)
sendfile: (The SendFile processing software library version)
alert: (The Alert processing software library version)
dbsync: (The DBSync processing software library version)
} (end software.engine):
{ software.library: (the library software subset)
minilibrary: (The Mini-Library (core) release version)
networking: (The Networking library release version)
reader: (The Reader Command library release version)
} (end software.library):
{ software.feature: (the feature software subset)
camera: (The Camera library release version)
scale: (The Scale library release version)
gps: (The GPS library release version)
lock: (The Lock library release version)
battery: (The Mobile Battery library release version)
} (end software.feature):
} (end software):
reader_int: (for each network interface on the reader, this will contain an array of it's settings. Each array includes:
name
sid (if wireless)
speed
duplex (if wired)
ipaddr
mask
gateway (if any)
dhcp
This array set is repeated for each interface on the reader.)
reader_status_access: (the date/time this was generated)
}

Notes:

- While this information is availabe from the Cloud Service, it will report the 'last reported' copy of the information sent from the readers to the Cloud Servers, which, if the Cloud Sync is not running, may not be current information.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       tagnumb: myTagNumber
       };


$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/get_reader_status’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     // System Information
                     myReaderID = r.results.system.reader_id;
                     myReaderModel = r.results.system.reader_model;
                     myReaderName = r.results.system.name;
                     myReaderGroupName = r.results.system.groupname;
                     myReaderModel = r.results.system.model;
                     myReaderMemory = r.results.system.memory;
                     myReaderStorage = r.results.system.storage;
                     myReaderOpMode = r.results.system.reader_op_mode;
                     myReaderManage = r.results.system.reader_manage_by;
                     myReaderAlertsEnabled = r.results.system.reader_alerts_enabled;
                     myReaderDBSyncEnabled = r.results.system.reader_dbsync_enabled;
                     myReaderAPIEnabled = r.results.system.api_enabled;
                     myReaderRemoteAddr = r.results.system.remoteaddr;
                     // Reader Specific Information
                     myReaderIP = r.results.reader.reader_ip;
                     myReaderMode = r.results.reader.reader_mode;
                     myReaderIsAlive = r.results.reader.reader_is_alive;
                     myReaderIsRunning = r.results.reader.reader_is_running;
                     myReaderAntNum = r.results.reader.ant_num;
                     myReaderAntPower = r.results.reader.ant_power;
                     myReaderAntRegion = r.results.reader.ant_region;
                     myReaderLoction = r.results.reader.location;
                     myReaderZone = r.results.reader.zones;
                     myReaderSubZones = r.results.reader.subzones;

                     SubZones = JSON.parse(myReaderSubZones);

                     myReaderFeatures = r.results.reader.enabled_features;
                     myReaderLastUpd = r.results.reader.last_update; // cloud only
                     myReaderLastUpdUTC = r.results.reader.last_update_utc; // cloud only
                     // Reader Software Information - Core
                     myReaderCoreOSVer = r.results.software.core.os;
                     myReaderCoreZMVer = r.results.software.core.zonemanager;
```

```
                        myReaderCoreBEVer = r.results.software.core.backend;
                        // Reader Software Information - Engine
                        myReaderEngineLic = r.results.software.engine.licensing;
                        myReaderEngineAPI = r.results.software.engine.api;
                        myReaderEngineDB = r.results.software.engine.database;
                        myReaderEngineSF = r.results.software.engine.sendfile;
                        myReaderEngineAlert = r.results.software.engine.alert;
                        myReaderEngineDBSync = r.results.software.engine.dbSync;
                        // Reader Software Information - Libraries
                        myReaderLibMini = r.results.software.library.mainilibrary;
                        myReaderLibAPI = r.results.software.library.api;
                        myReaderLibNet = r.results.software.library.networking;
                        myReaderLibReader = r.results.software.library.reader;
                        // Reader Software Information - Features
                        myReaderFeatCamera = r.results.software.feature.camera;
                        myReaderFeatScale = r.results.software.feature.scale;
                        myReaderFeatGPS = r.results.software.feature.gps;
                        // Interface Information
                        $.each(r.results.reader_int, function(k,v) {
                              console.log('Name: '+v.name);
                              console.log('SID: '+v.sid);
                              console.log('Speed: '+v.speed);
                              console.log('Duplex: '+v.duplex);
                              console.log('IPAddr: '+v.ipaddr);
                              console.log('SubnetMask: '+v.mask);
                              console.log('Gateway: '+v.gateway);
                              console.log('DHCP Enabled: '+v.dhcp);
                        });
                  };
            }
});
```

Alert Control APIs

start_alert_engine

This call will start the alert background monitoring process. You must start the background monitoring process before any alerts will work.

Synopsis:

start_alert_engine
Input:

sessionkey:	(an active Sessionkey)

Output:

request:	'start_alert_engine'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.
- This feature is currently not available in the Cloud Service as the alert engine is always enabled.

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/start_alert_engine',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              engineEnabled = r.success;
       }
});
```

stop_alert_engine

This call will stop the alert background monitoring process.

Synopsis:

stop_alert_engine
Input:

sessionkey:	(an active Sessionkey)

Output:

request:	'stop_alert_engine'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.
- This feature is currently not available in the Cloud Service as the alert engine is always enabled.

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
	url: 'https://(remote_site)/rfrainapi.php/stop_alert_engine',
	dataType: 'json',
	type: 'POST',
	data: myData,
	success: function(r) {
		engineStopped = r.success;
	}
});
```

define_alert

This call will create an alert entry with the given options. The options are compared to active tags as they are scanned (or in the case of the Cloud service, as the data arrives in the Cloud) to see if they match the specified criteria. For detailed explanations of the criterias, and how the alerts work, please see the ZoneManager documentation. Alerts may be created on the SmartReader series, Gateway, Enterprise, or Cloud Services.

Synopsis:

define_alert

Input:

sessionkey:	(an active Sessionkey)
dbname:	(database name, the default is RfrainDB {note: this option only applies to the SmartReader series, and may be left blank for other services})
alertopts:	(The alert definition options)

Output:

request:	'define_alert'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	
{	
alertid:	The Internal ID of this alert
}	

Notes:

- The alertid is returned from this call if created successfully. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.
- The dbname parameter only applies to the SmartReader series. (On the SmartReaders series, this is always 'RfrainDB').
- This call will create an alert. To activate that alert, use the 'start_alert' API call with the returned 'alertid'.
- The alertopts are specified in industry standard JSON format as '{ name:value, name:value, ... }' sets of pairs.
- Alert Definition Options (alertopts):

define_alert ctd.

PARAM	REQUIRED	DESCRIPTION
name	Yes	The name of this alert
tagnumb	Yes	The tag number this alert will monitor, or an "*" for all tags
mode	Yes	The alert mode, one of: subzone, event
type	Yes	The type of alert, one of: history, or jitter
notifygroup	Yes	The alert notify group to receive this notice
comwhat	Cond	Compare What, for use with "event" alerts, and will be detectstat
comcond	Cond	Comparison, for use with "event" alerts, and will be equal
comto	Cond	Compare To What Value, for use with "event" alerts, and will be one of PRES, MISS, or IN
gpio	Cond	For use with "gpio" alerts, and will be gpio port number to fire against
sec_comp		(optional) Secondary Alert Comparison: field to check, (one of tagname, custom1 - custom5)
sec_cond		(optional) Secondary Alert Comparison: comparison, (one of equal, notequal)
sec_val		(optional) Secondary Alert Comparison: value to compare for (string)

The "Cond" conditional parameters are only required on certain alert definitons.

Example 1: Create a SubZone Change Alert: "alertopts" passed:

{"name":"myAlert1","tagnumb":"00-01-02-03-04-05-06-07-08-09-10-11","mode":"subzone","type":"history","notifygroup":"1"}

This would create a "subzone" alert monitoring the specified tagnumber. If that tag should change subzones, or appear in a new subzone, an alert would be triggered.

Example 2: Create an event based alert: "alertopts" passed:

{"name":"myAlert2","tagnumb":"00-01-02-03-04-05-06-07-08-09-10-12","mode":"event","type":"jitter","notifygroup":"1","compwhat":"detectstat","compcond":"equal","compwhatto":"MISS"}

This would create an alert against the specified tagnumber what would fire an alert to notifygroup 1 when the tag went missing but using the jitter events (suppressed rapid changes).

define_alert ctd.

Example 3: Create a SubZone Change Alert for All Tags: “alertopts” passed:

{“name”:”myAlert3”,”tagnumb”:”*”,”mode”:”subzone”,”type”:”jitter”,”notifygroup”:”1”}

This would create a “subzone” alert monitoring the specified tagnumber using the jitter events (suppressed rapid changes). If any tag should change subzones, or appear in a new subzone, an alert would be triggered.

The “Cond” conditional parameters are only required on certain alert definitons.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       dbname: myDBName,
       alertopts: myAlertDefinition,
 };

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/define_alert’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              alertCreated = r.success;
              alertid = r.results.alertid;
       }
});
```

start_alert

This call will start an alert that has been defined through the ZoneManager Web Interface, or through the "create_alert" API call. To use the call, the 'alertid' must be specified. This can be found using the "list_alerts" API call.

Synopsis:

start_alert

Input:

sessionkey:	(an active Sessionkey)
alertid:	(the uique ID of the alert as assigned by the system)

Output:

request:	'start_alert'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       alertid: myAlertID
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/start_alert’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { alertEnabled = r.success; }
});
```

stop_alert

This call will stop an active alert that has been defined through the ZoneManager Web Interface, or through the "create_alert" API call. To use the call, the 'alertid' must be specified. This can be found using the "list_alerts" API call.

Synopsis:

stop_alert

Input:

sessionkey:	(an active Sessionkey)
alertid:	(the uique ID of the alert as assigned by the system)

Output:

request:	'stop_alert'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       alertid: myAlertID
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/stop_alert’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { alertEnabled = r.success; }
});
```

start_all_alerts

This call will start all active alerts that has been defined through the ZoneManager Web Interface, or through the "create_alert" API call.

Synopsis:

start_all_alerts

Input:
sessionkey: (an active Sessionkey)

Output:
request: 'start_all_alerts'
success: TRUE/FALSE
message: (api call detailed result or error message)
results: { }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/start_all_alerts',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { alertsStarted = r.success; }
});
```

stop_all_alerts

There are times when you may simply need to turn off all active alerts. This call will stop all active alerts that has been defined through the ZoneManager Web Interface, or through the "create_alert" API call.

Synopsis:

stop_all_alerts

Input:
sessionkey: (an active Sessionkey)

Output:
request: 'stop_all_alerts'
success: TRUE/FALSE
message: (api call detailed result or error message)
results: { }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/stop_all_alerts',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { alertsDisabled = r.success; }
});
```

delete_alert

This call will delete an alert that has been defined through the ZoneManager Web Interface, or through the "create_alert" API call. To use the call, the 'alertid' must be specified. This can be found using the "list_alerts" API call.

Synopsis:

delete_alert

Input:

sessionkey:	(an active Sessionkey)
alertid:	(the unique ID of the alert as assigned by the system)

Output:

request:	'delete_alert'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       alertid: myAlertID
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/delete_alert’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { alertDeleted = r.success; }
});
```

list_alerts

This call will return the list of alerts currently defined on the system. Returned data includes the unqiueID assigned to the alert and the alert settings in the same form as the "create_ alert" API call uses to create an alert.

Synopsis:

list_alerts

Input:

sessionkey:	(an active Sessionkey)

Output:

request:	'list_alerts'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	

```
{
  {
```

alertid:	(the uniqueID assigned by the system to this alert)
name:	(the name of this alert)
tagnumb:	(the tag number this alert is monitoring)
active:	(if the alert is active)
amode:	(the alert mode)
alertagainst:	(the type of this alert, history or jitter)
asign:	(the alert condition comparison sign)
alerttable:	(the table this alert is checking)
alertcol:	(the column this alert is checking)
alertgpio:	(the gpio this alert is checking {if on a smartreader advanced})
timeinterval:	(the repeat notification interval if this is a repeating {notify} alert)
notifygroup:	(the notigy group to receive this alert notification)

```
  }
...
}
```

Notes:

- A special license is required to use the alert features.

---

list_alerts ctd.

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
	url: 'https://(remote_site)/rfrainapi.php/list_alerts',
	dataType: 'json',
	type: 'POST',
	data: myData,
	success: function(r) {
		if(r.success) {
			$.each(r.results, function(k,v) {
				myAlertID = v.alertid;
				myAlertName = v.name;
				myAlertTag = v.tagnumb;
				myAlertMode = v.amode;
				myAlertCol = v.acol;
				myAlertSign = v.asign;
				myAlertVal = v.aval;
				myAlertTime = v.timeinterval;
			});
		};
	}
});
```

define_alert_group

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call allows you to create a notification group that can then be associated with an alert.

Synopsis:

define_alert_group
Input:

sessionkey:	(an active Sessionkey)
agroupname:	(The Name of the New Group)

Output:

request:	'define_alert_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
                sessionkey: myKey,
                agroupname: myGroupName
         };

$.ajax({
        url: 'https://(remote_site)/rfrainapi.php/define_alert_group',
        dataType: 'json',
        type: 'POST',
        data: myData,
        success: function(r) { AlertGroupCreated = r.success; }
});
```

delete_alert_group

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call allows you to delete a notification group. You will be provided with a list of alerts that were using that group.

Synopsis:

delete_alert_group

Input:

sessionkey:	(an active Sessionkey)
agroupid:	(The ID of the Alert Notify Group)

Output:

request:	'delete_alert_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- If any alerts where using only the deleted group, you will need to update those alerts to another group destination
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
                sessionkey: myKey,
                agroupname: myGroupName
         };

$.ajax({
        url: 'https://(remote_site)/rfrainapi.php/delete_alert_group',
        dataType: 'json',
        type: 'POST',
        data: myData,
        success: function(r) {
                if(r.success) {
                        myListOfAlertsAffected = r.results.alerts_affected;
                };
        }
});
```

list_alert_groups

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call will list all defined notification groups.

Synopsis:

list_alert_groups

Input:

sessionkey:	(an active Sessionkey)

Output:

request:	'list_alert_groups'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	

```
{
 {
```

groupid:	(Alert Notification Group ID)
groupname:	(Alert Notification Group Name)
email:	(A List of Defined Email EndPoints in this Group)
api:	(A List of Defined API EndPoints in this Group)

```
 }
...
}
```

Notes:

- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = { sessionkey: myKey };

$.ajax({
        url: 'https://(remote_site)/rfrainapi.php/list_alert_groups',
        dataType: 'json',
        type: 'POST',
        data: myData,
        success: function(r) {
                if(r.success) {
                        $.each(r.results, function(k,v) {
                                myAlertGroupID = v.groupid;
                                myAlertGroupName = v.groupname;
                                myAlertGroupEmails = v.email;
                                myAlertGroupApis = v.api;
                        });
                };
        }
});
```

add_alert_email_entry_to_group

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call will allow you to add an email contact to a defined alert group.

Synopsis:

add_alert_email_entry_to_group
Input:

sessionkey:	(an active Sessionkey)
email:	(a valid email address to receive alerts for this group)
agroupid:	(an active alert groupid)

Output:

request:	'add_alert_email_entry_to_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       email: myEmail,
       agroupid: myAGroupID,
       };

$.ajax({
       url:
'https://(remote_site)/rfrainapi.php/add_alert_email_entry_to_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { myEmailWasAdded = r.success; };
});
```

remove_alert_email_entry_from_group

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call will allow you to remove an email contact from a defined alert group.

Synopsis:

remove_alert_email_entry_from_group
Input:

sessionkey:	(an active Sessionkey)
email:	(an email address to be removed from this group)
agroupid:	(an active alert groupid)

Output:

request:	'remove_alert_email_entry_from_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       email: myEmail,
       agroupid: myAGroupID,
       };

$.ajax({
       url:'https://(remote_site)/rfrainapi.php/remove_alert_email_entry_from_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { myEmailWasRemoved = r.success; };
});
```

add_alert_api_entry_to_group

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call will allow you to add an API contact to a defined alert group.

Synopsis:

add_alert_api_entry_to_group
Input:

sessionkey:	(an active Sessionkey)
api:	(a valid api to receive alerts for this group)
agroupid:	(an active alert groupid)

Output:

request:	'add_alert_api_entry_to_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       api: myAPI,
       agroupid: myAGroupID,
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/add_alert_api_entry_to_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { myAPIWasAdded = r.success; };
});
```

remove_alert_api_entry_from_group

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call will allow you to remove an API receiver from a defined alert group

---

Synopsis:

remove_alert_api_entry_from_group
Input:

sessionkey:	(an active Sessionkey)
api:	(an api receiver to be removed from this group)
agroupid:	(an active alert groupid)

Output:

request:	'remove_alert_api_entry_from_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       api: myAPI,
       agroupid: myAGroupID,
       };

$.ajax({
       url:
'https://(remote_site)/rfrainapi.php/remove_alert_api_entry_from_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { myAPIWasRemoved = r.success; };
});
```

add_notify_group_to_alert

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call will allow you to add a defined alert group to a defined alert.

Synopsis:

add_notify_group_to_alert
Input:

sessionkey:	(an active Sessionkey)
alertid:	(an active alertid)
agroupid:	(an active alert groupid)

Output:

request:	'add_notify_group_to_alert'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       alertid: myAlertID,
       agroupid: myAGroupID,
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/add_notify_group_to_alert',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { myGroupWasAdded = r.success; };
});
```

remove_notify_group_from_alert

Alerts in the Gateway, Enterprise, or Cloud services allow for notification groups. These groups can consist of email or api notification endpoints. This call will allow you to remove a notify group from an alert.

Synopsis:

remove_notify_group_from_alert
Input:

sessionkey:	(an active Sessionkey)
alertid:	(an active alertid)
agroupid:	(an active alert groupid)

Output:

request:	'remove_notify_group_from_alert'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.
- A special license is required to use the alert features.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       alertid: myAlertID,
       agroupid: myAGroupID,
       };

$.ajax({
       url:'https://(remote_site)/rfrainapi.php/remove_notify_group_from_alert',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { myGroupWasRemoved = r.success; };
});
```

System Control APIs

get_system_users

This call allows you to get all system users for the company.

Synopsis:

get_system_users
Input:
sessionkey: (an active Sessionkey)
cname: (the company name)

Output:
category: system_api
request: 'get_system_users'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:
{
email: user email address
username: user name
active: status Active=Y/N
usertype: user level, admin or user
}

Notes:

- As the calling user, you must have admin level access to use this API call
- Level can be either 'user' or 'admin'
- Results will show email, username, active=Y/N, usertype

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
	sessionkey: myKey,
	cname: myCompany
};
$.ajax({
	url: 'https://(remote_site)/rfrainapi.php/get_system_users',
	dataType: 'json',
	type: 'POST',
	data: myData,
	success: function(r) {
		if(r.success) {
			myUserEmail = r.results.email;
			myUserName = r.results.username;
			myUserActive = r.results.active;
			myUserType = r.results.usertype;
		};
	}
});
```

add_user

This call allows you to add a User to the RFRain SmartReader, Enterprise Server or Cloud Service. The user will have API Access at the level you decide, and the local WebInterface at the level you decide.

Synopsis:

add_user

Input:

sessionkey:	(an active Sessionkey)
login:	(the new user's Login (Email Address). This must be Base64 Encoded before transmission)
level:	(the new user's access level)
passwd:	(the new user's password. This must be Base64 Encoded before transmission)

Output:

request:	'add_user'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- As the calling user, you must have admin level access to use this API call
- Level can be either 'user' or 'admin'
- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       login: btoa(myUserID),
       level: myUserLevel,
       passwd: btoa(myGoodPassword)
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/add_user’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { userAdded = r.success; };
});
```

delete_user

This call allows you to remove a User to the RFRain SmartReader, Enterprise Server or Cloud Service. The user will no longer have access.

Synopsis:

delete_user

Input:

sessionkey:	(an active Sessionkey)
login:	(the user's Login (Email Address). This must be Base64 Encoded before transmission)

Output:

request:	'delete_user'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- As the calling user, you must have admin level access to use this API call
- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       login: btoa(myUserID),
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/delete_user’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { userDeleted = r.success; };
});
```

change_user_passwd

This call allows you to change a User's password on the RFRain SmartReader, Enterprise Server or Cloud Service.

Synopsis:

change_user_passwd

Input:

sessionkey:	(an active Sessionkey)
login:	(the user's Login (Email Address). This must be Base64 Encoded before transmission)
passwd:	(the user's new password. This must be Base64 Encoded before transmission)
passwdc:	(the user's new password again for verification. This must be Base64 Encoded before transmission)

Output:

request:	'change_user_passwd'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- As the calling user, you must have admin level access to use this API call
- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       login: btoa(myUserID),
       passwd: btoa(myNewPassword),
       passwd: btoa(myNewPassword)
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/change_user_passwd’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { userPasswdChanged = r.success; };
});
```

change_user_level

This call allows you to change a User's access level on the RFRain SmartReader, Enterprise Server or Cloud Service.

Synopsis:

change_user_level

Input:

sessionkey:	(an active Sessionkey)
login:	(the user's Login (Email Address). This must be Base64 Encoded before transmission)
level:	(the user's new level)

Output:

request:	'change_user_level'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- As the calling user, you must have admin level access to use this API call
- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       login: btoa(myUserID),
       level: myNewLevel
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/change_user_level’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { userLevelChanged = r.success; };
});
```

Feature Control APIs

camera_set_retention

This call allows you to set the image retention policy for the Camera feature. There are several retention levels you may choose from. These levels allow you to control space usage and fresensic retention policies.

Synopsis:

camera_set_retention

Input:

sessionkey:	(an active Sessionkey)
camid:	(which camera to start monitoring)
ret:	(the retention policy)

Output:

request:	'camera_set_retention'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- As the calling user, you must have admin level access to use this API call
- As of this release, the SmartReader only supports up to four cameras. The CameraID number will be between 1 and 4.
- A special license is required to use the camera feature.
- Controlling image retention is vital to system performance. Too many, and the system will run out of space. Too few, and you do not have enough images saved for proper forensics and analysis.
- No results are returned from this call. The 'success' parameter will indicate whether or not the call was successfull.

camera_set_retention continued

- Smart Reader Retention Values (Smart Reader Only)

RETENTION	VALUE	DESCRIPTION
space	70% Avail Storage	This will being to delete the oldest images when 70% of available system storage space is reached. This is system default. NOTE: No matter the selections below, the space limit is ALWAYS enforced to prevent space exauhstion problems.
img1	100 Images	This will being to delete the oldest images when the number of images exceeds 100.
img2	300 Images	This will being to delete the oldest images when the number of images exceeds 300.
img3	600 Images	This will being to delete the oldest images when the number of images exceeds 600.
img4	1,200 Images	This will being to delete the oldest images when the number of images exceeds 1,200.
img5	3,000 Images	This will being to delete the oldest images when the number of images exceeds 3,000.
time1	1 week	This will being to delete the oldest images when their age exceeds 1 week.
time2	2 weeks	This will being to delete the oldest images when their age exceeds 2 weeks.
time3	1 month	This will being to delete the oldest images when their age exceeds 1 month
time4	6 months	This will being to delete the oldest images when their age exceeds 6 months.

- Smart Reader Retention Values (Smart Reader Only)

RETENTION	VALUE	DESCRIPTION
c_space	90% Avail Storage	This will being to delete the oldest images when 90% of available system storage space is reached. This is system default. NOTE: No matter the selections below, the space limit is ALWAYS enforced to prevent space exauhstion problems.
c_img1	600 Images	This will being to delete the oldest images when the number of images exceeds 600.
c_img2	1,200 Images	This will being to delete the oldest images when the number of images exceeds 1,200.
c_img3	3,000 Images	This will being to delete the oldest images when the number of images exceeds 3,000.
c_img4	6,000 Images	This will being to delete the oldest images when the number of images exceeds 6,000.
c_img5	12,000 Images	This will being to delete the oldest images when the number of images exceeds 12,000.
c_time1	1 month	This will being to delete the oldest images when their age exceeds 1 month
c_time2	4 months	This will being to delete the oldest images when their age exceeds 4 months
c_time3	6 months	This will being to delete the oldest images when their age exceeds 6 months
c_time4	1 year	This will being to delete the oldest images when their age exceeds 1 year.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       camid: myCamera,
       ret: myRetention
};

$.ajax({
       url: ‘https://(remote_site)/rfrainapi.php/camera_set_retention’,
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { cameraRetentionSet = r.success; };
});
```

camera_event_association

FEATURE: Given an event UTC timestamp, this call allows you to see if any images from the Camera feature fall within the scope of that event. This allows you to associate an image capture with a tag event.

Synopsis:
camera_event_association

Input:
sessionkey: (an active Sessionkey)
camid: (optional: the physical camera id from reader)
tagnumb: (optional: a tagnumber to check against)
readername: (optional: the reader identity readername)
groupname: (optional: the reader identity groupname)
timestamp: (the UTC Timestamp around which to search)
iassoc: (the range around the timestamp which to search, in sec.)

Output:
request: 'camera_event_association'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:
{
{
imageid: (the internal image id)
readerid: (the readerid that sourced this image)
readername: (the identity name of the reader that sourced this image)
groupname: (the identity groupname that sourced this image)
access: (the date/timestamp of the image)
accessutc: (the UTC timestamp of the image)
imagesize: (the size of the image)
framecnt: (the number of image frames in the image)
imglength: (the number of seconds of footage in the image)
imageurl: (a URL to an image that is associated with that timestamp event)
taglist: (a repeating array of [tagnumb, tagname, subzone, access, accessutc, lat, lng] that matches the specified criteria)
}
...
}

Notes:

- The image range of capture falls within the range (timestamp - iassoc) to (timestamp + iassoc). If an image covers or falls within that time frame and matches the other criteria, it will be returned for inspection.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
	sessionkey: myKey,
	readername: myReaderName
	groupname: myGroupName
	timestamp: myDateTime_or_UTC_ToCheck
	iassoc: myIAssoc
};

$.ajax({
	url: ‘https://(remote_site)/rfrainapi.php/camera_event_association’,
	dataType: 'json',
	type: 'POST',
	data: myData,
	success: function(r) {
		if(r.success) {
			$.each(r.results, function(k,v) {
				MyImageID = v.imageid;
				MyImageSourceReaderID = v.readerid;
				MyImageSourceReaderName = v.readername;
				MyImageSourceGroupName = v.groupname;
				MyImageAccess = v.access;
				MyImageAccessUTC = v.accessutc;
				MyImageSize = v.imagesize;
				MyImageFrameCnt = v.framecnt;
				MyImageLength = v.imglength;
				MyImageURL = v.imgurl;
				MyListOfTags = v.taglist;
			});
		};
	}
});
```

Cloud Specific APIs

define_reader_name

This call will define reader name.

Synopsis:

define_reader_name
Input:

sessionkey:	(an active Sessionkey)
readerid:	(reader id, default is 'RfrainReader')
readername:	(reader name, required field)

Output:

category:	cloud_api
request:	'define_reader_name'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       readerid: myReader
       readername: myReaderName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/define_reader_name',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { }
});
```

define_reader_group

This call will define new reader group

Synopsis:

define_reader_group
Input:
sessionkey: (an active Sessionkey)
groupname: (group name, required field)

Output:
category: cloud_api
request: 'define_reader_group'
success: TRUE/FALSE
message: (api call detailed result or error message)
results: { }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       groupname: myGroupName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/define_reader_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { }
});
```

delete_reader_group

This call will delete reader group.

Synopsis:

delete_reader_group
Input:
sessionkey: (an active Sessionkey)
groupname: (group name, required field)

Output:
category: cloud_api
request: 'delete_reader_group'
success: TRUE/FALSE
message: (api call detailed result or error message)
results: { }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       groupname: myGroupName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/delete_reader_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { }
});
```

list_reader_groups

This call will display all groups.

Synopsis:

list_reader_groups
Input:
sessionkey: (an active Sessionkey)

Output:
category: cloud_api
request: 'list_reader_groups'
success: TRUE/FALSE
message: (api call detailed result or error message)
results:
{
0: groupname1
1: groupname2
2: groupname3
}

Notes:

- Results will display group names.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/list_reader_groups',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myGroupName = v;
                     };
              };
       }
});
```

add_reader_to_group

This call will add the reader in group.

Synopsis:

add_reader_to_group
Input:

sessionkey:	(an active Sessionkey)
readerid:	(reader id, default is 'RfrainReader')
groupname:	(groupname where reader is to be added), required field

Output:

category:	cloud_api
request:	'add_reader_to_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not

---

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       readerid: myReader
       groupname: myGroupName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/add_reader_to_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { }
});
```

remove_reader_from_group

This call will remove reader from specified group.

Synopsis:

remove_reader_from_group
Input:

sessionkey:	(an active Sessionkey)
readerid:	(reader id, default is 'RfrainReader')
groupname:	(group name, required field)

Output:

category:	cloud_api
request:	'remove_reader_from_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- No results are returned from this call, the 'success' field will tell you if it was successful or not

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       readerid: myReader
       groupname: myGroupName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/remove_reader_from_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) { }
});
```

list_all_readers_by_group

This call will display all readers within different groups

Synopsis:

list_all_readers_by_group
Input:

sessionkey:	(an active Sessionkey)

Output:

category:	cloud_api
request:	'list_all_readers_by_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- All readers within each group will be displayed with readerid, readername and group name.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/list_all_readers_by_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myReader = r.results.readerid;
                            myReaderName = r.results.readername;
                            myGroupName = r.results.groupname;
                     };
              };
       }
});
```

list_readers_in_group

This call will display all readers within specified group.

Synopsis:

list_readers_in_group
Input:

sessionkey:	(an active Sessionkey)
groupname:	(group name, required field)

Output:

category:	cloud_api
request:	'list_readers_in_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- Results will display readerid, reader name, group name.

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       groupname: myGroupName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/list_readers_in_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myReader = r.results.readerid;
                            myReaderName = r.results.readername;
                            myGroupName = r.results.groupname;
                     };
              };
       }
});
```

find_group_reader_is_in

This call will find group that reader belongs to.

Synopsis:

find_group_reader_is_in
Input:

sessionkey:	(an active Sessionkey)
readerid:	(reader id, required field)

Output:

category:	cloud_api
request:	'find_group_reader_is_in'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- Results will show readerid, readername and groupname

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       readerid: myReaderName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/find_group_reader_is_in',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myReader = r.results.readerid;
                            myReaderName = r.results.readername;
                            myGroupName = r.results.groupname;
                     };
              };
       }
});
```

get_reader_status_by_group

This call will show reader status within specified group.

Synopsis:

get_reader_status_by_group
Input:

sessionkey:	(an active Sessionkey)
groupname:	(group name, required field)

Output:

category:	cloud_api
request:	'get_reader_status_by_group'
success:	TRUE/FALSE
message:	(api call detailed result or error message)
results:	{ }

Notes:

- Results will show all details of reader status, system status, software status, interfaces for each reader within specified group

Example (JavaScript, JQuery style AJAX call):

```
var myData = {
       sessionkey: myKey,
       groupname: myGroupName
       };

$.ajax({
       url: 'https://(remote_site)/rfrainapi.php/get_reader_status_by_group',
       dataType: 'json',
       type: 'POST',
       data: myData,
       success: function(r) {
              if(r.success) {
                     $.each(r.results, function(k,v) {
                            myStatusVersion = r.results.status_version;
                            myAccess = r.results.access;
                            $.each(r.results.system, function(k,v) {
                                   mySystemReaderName = r.results.system.readername;
                                   mySystemGroupName = r.results.system.groupname;
                                   mySystemModel = r.results.system.model;
                                   mySystemMemory = r.results.system.memory;
                                   mySystemStorage = r.results.system.storage;
```

```
                    mySystemOpMode = r.results.system.op_mode;
                    mySystemManagedBy = r.results.system.manage_by;
                    mySystemAlertsEnabled = r.results.system.alerts_enabled;
                    mySystemDbSyncEnabled = r.results.system.dbsync_enabled;
                    mySystemApiEnabled = r.results.system.api_enabled;
                    mySystemRemoteAddr = r.results.system.remote_addr;
                    mySystemLastUpdate = r.results.system.last_update;
                    mySystemLastUpdateUTC = r.results.system.last_update_utc;
                };
                $.each(r.results.reader, function(k,v) {
                    myReaderReaderIP = r.results.reader.reader_ip;
                    myReaderReaderMode = r.results.reader.reader_mode;
                    myReaderIsAlive = r.results.reader.is_alive;
                    myReaderIsRunning = r.results.reader.is_running;
                    myReaderAntNum = r.results.reader.ant_num;
                    myReaderAntPower = r.results.reader.ant_power;
                    myReaderAntRegion = r.results.reader.ant_region;
                    myReaderLocation = r.results.reader.location;
                    myReaderZone = r.results.reader.zone;
                    myReaderSubzones = r.results.reader.subzones;
                    myReaderEnabledFeatures = r.results.reader.enabled_
                    features;
                    myReaderLastUpdate = r.results.reader.last_update;
                    myReaderLastUpdateUTC = r.results.reader.last_update_utc;
                };
                $.each(r.results.software, function(k,v) {
                    mySoftwareCore = r.results.software.core;
                    mySoftwareEngine = r.results.software.engine;
                    mySoftwareLibrary = r.results.software.library;
                    mySoftwareFeature = r.results.software.feature;
                    mySoftwareStatus = r.results.software.status;
                };
                $.each(r.results.interfaces, function(k,v) {
                    myInterfacesReaderid = r.results.interfaces.readerid;
                    myInterfacesEnabled = r.results.interfaces.enabled;
                    myInterfacesName = r.results.interfaces.name;
                    myInterfacesSid = r.results.interfaces.sid;
                    myInterfacesSpeed = r.results.interfaces.speed;
                    myInterfacesDuplex = r.results.interfaces.duplex;
                    myInterfacesIPAddrv4 = r.results.interfaces.ipaddrv4;
                    myInterfacesMaskv4 = r.results.interfaces.maskv4;
                    myInterfacesIPAddrv6 = r.results.interfaces.ipaddrv6;
                    myInterfacesMaskv6 = r.results.interfaces.maskv6;
                    myInterfacesGateway = r.results.interfaces.gateway;
                    myInterfacesDHCP = r.results.interfaces.dhcp;
                    myInterfacesAccess = r.results.interfaces.access;
                };
            };
        };
    }
});
```



For more support email us at sales@rfrain.com
or call 1.833.273.7246

Email: sales@rfrain.com
Phone: 1-833-273-7246
Availability: 24/7
www.rfrain.com