Radius Web Services - Table of Contents

Page numbers are clickable in Word format only. PDF clicking is not supported.

Version Numbers

All versions are named using the following format: *MMDDYY*

Version 110522 – Released February 2023

- Added support for Printing Registration PDF through Webservices

Version 092421 – Released September 24, 2021

- Added support for Contact Engagements and Contact Engagements Logs, including Enhanced Searching.
- Added support for Campaigns, Communication Plans and Communication Plans Contacts, including Enhanced Searching.
- Added new method getStatistics, to retrieve Statistics for Campaigns and Communication Plans.

Version 030918 – Released March 09, 2018

- Added support for Test Scores, including Enhanced Searching.
- Added support for Export Filters.
- For “List All Modules” Web Service, added support to show system module name alongside the custom module name (display label).

Version 032417 – Released March 24, 2017

- Added support for Date, DateTime, Month/Year, & Multi-select searches to the “Enhanced Search for Entities” capabilities.

Version 020317 – Released February 03, 2017

- Added “Enhanced Search for Entities” capabilities.

Version 072916 – Released July 29, 2016 •
Added hosts for STAGING environments.

Version 062416 – Released June 24, 2016
- Added support for Requirements.
- Added Events Module Details – Recurring Events can be read via the Web Services, but cannot currently be modified.

Version 031116 – Released March 25, 2016
- Added support for Targets.

Version 121115 – Released December 11, 2015 • Added
support for Events and Attendees modules.
- Added Events, CaseMessages & Registrations module information to the "Module Specific Details" section.
- Added PHP Samples & Code.

Version 031615 – Released March 16, 2015
- Added detailed User & Inquiries module information to the "Module Specific Details" section.
- Added support for Decisions module.

Version 020615 – Released February 6, 2015
- Added support for FieldID's. See the "List All Fields for a Module" Web Service for details.
- Added ability to create Custom Fields.
- Added support for the Users, Groups, Roles, & Profiles modules.
- Added "Notes About Specific Modules" Section.
- Added "Digest Authentication" Resource.
- Added "Code Samples" Section.

Version 120414 – Released December 4, 2014
- Added support for "newerThan" and "updatedSince" for searches.

Version 110514 – Released November 5, 2014

Version 101714 – Released October 17, 2014

Version 090814 – Released September 8, 2014

RADIUS WEB SERVICES

Summary

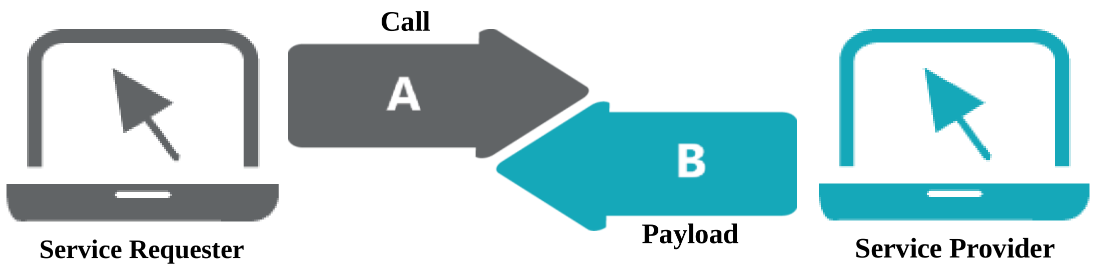
Web Services gives Radius clients the ability to customize and launch a myriad of data transfers with Radius by utilizing the Web Service API methods documented within.

Clients will need staff on hand with Information Technology (IT) capabilities to utilize the API methods made available via Web Services. The detailed documentation that follows provides your IT staff with the appropriate information and guidelines for implementing the Radius Web Services.

Please keep in mind that you may have data transfer and integration needs where Web Services may not be the best fit. These include scenarios such as transferring Personally Identifiable Information, complex Student Information System integrations, batch file transfers, data transformations, etc.

What is a Web Service?

At its most fundamental level, a web service is a mode of communication that facilitates the interaction of two machines over a network. In a web services transaction there are two essential parties, the *service requester* and the *service provider*.



This illustration depicts the transfer of data between the **Service Requester** and the **Service Provider**. In this transaction the service requester places a **Call** to the service provider. Upon submitting a successful call, the service provider will return the **Payload.** Radius serves as the service provider in this transaction and can accept Web Service requests from numerous types of service requesters, such as a Student Information System or lead generation service. Regardless of the specific requester, the setup and use processes remain the same.

The Components of a Web Service

There are few elements to a web service; however, these elements are essential for executing a successful transmission of data. Each web service transaction consists of:

- URL
- HTTP Method
- GET
- POST
- PUT
- DELETE
- Payload – If using POST, PUT, or DELETE

HTTP Methods

GET – The GET command is utilized to retrieve a resource. Potential resources include a list of system modules, fields, or records from a module.

POST – The POST command is utilized when creating a resource on the server. Potential resources include the creation of a contact record, organization record, or education record.

PUT – The PUT command is utilized when changing the state or updating a resource. Potential changes might include updating demographic information, and education records.

DELETE – The DELETE command is utilized when removing or deleting a resource.

Responses

All web service responses are in JavaScript Object Notation (JSON) format. A successful response will yield the following message.

```
{
     "status":"ok",
     "payload": <response data>
}
```

An unsuccessful response will yield the following message.

```
{
     "status":"error",
     "message": "<error message>"
}
```

HTTP Response Codes

- 200 – OK
- 201 – Created
- **400** – Bad Request
- **404** – Not Found
- **500** – Server Error

Radius Web Service Set-up

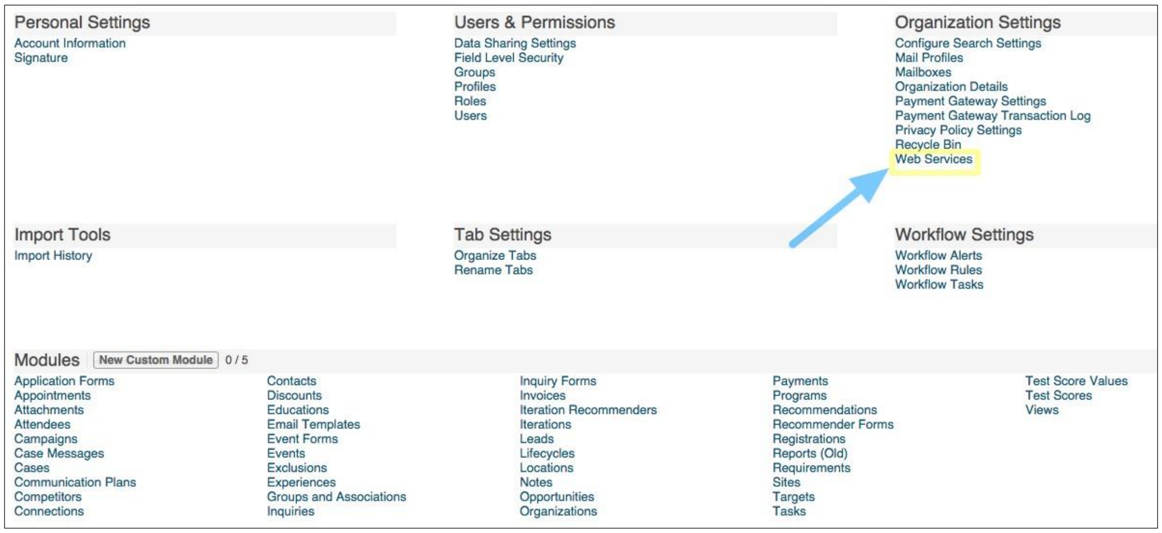
The Radius Web Service is built upon the Representational State Transfer (REST) architectural style, and utilizes JavaScript Object Notation (JSON) format.

The web service is a real-time exchange of data and is designed for the transmission of single records. Larger transfers should utilize an import/export process for efficiency.

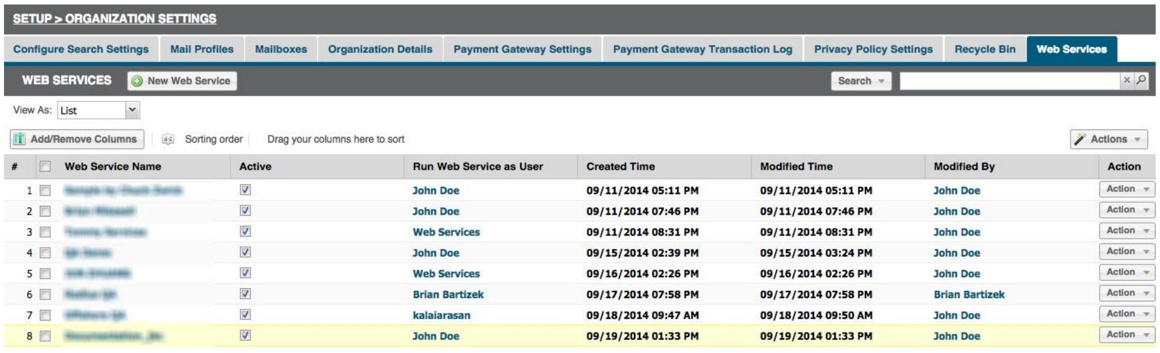
Creating a Web User Account

A Radius administrator can create any number of Web Services accounts. Each account is unique and provides the user with credentials that must be utilized to execute a successful web service transaction.

Step 1: Navigate to the Web Services
Navigate and click on **Setup**. The link can be found in the upper-right portion of the Radius user interface. **Web Services** is accessible via **Organization Settings.**

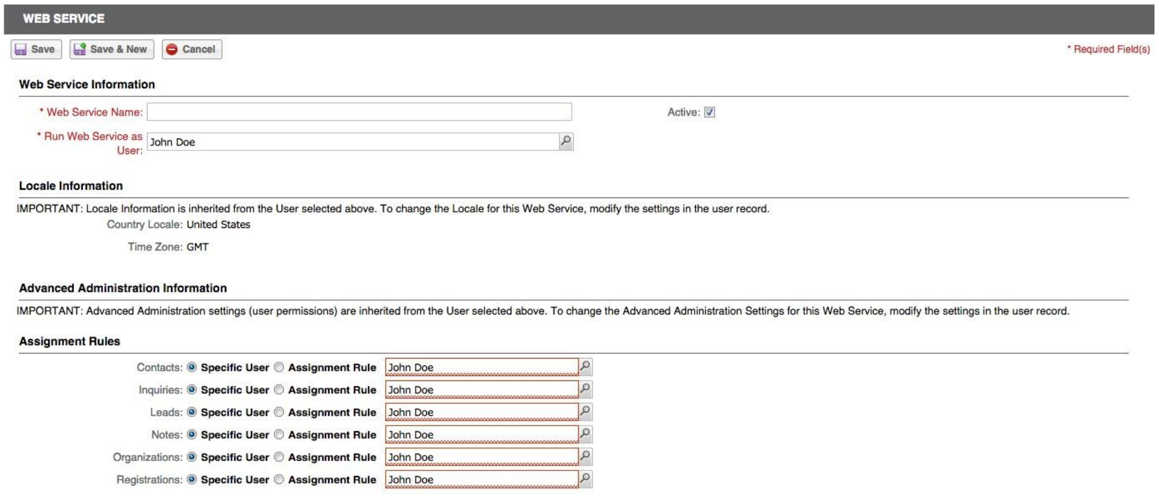


Upon entering the Web Service menu, the user is presented with a listing of all created accounts.



#		Web Service Name	Active	Run Web Service as User	Created Time	Modified Time	Modified By	Action
1	☐	[redacted]	☑	John Doe	09/11/2014 05:11 PM	09/11/2014 05:11 PM	John Doe	Action
2	☐	[redacted]	☑	John Doe	09/11/2014 07:46 PM	09/11/2014 07:46 PM	John Doe	Action
3	☐	[redacted]	☑	Web Services	09/11/2014 08:31 PM	09/11/2014 08:31 PM	John Doe	Action
4	☐	[redacted]	☑	John Doe	09/15/2014 02:39 PM	09/15/2014 03:24 PM	John Doe	Action
5	☐	[redacted]	☑	Web Services	09/16/2014 02:26 PM	09/16/2014 02:26 PM	John Doe	Action
6	☐	[redacted]	☑	Brian Bartizek	09/17/2014 07:58 PM	09/17/2014 07:58 PM	Brian Bartizek	Action
7	☐	[redacted]	☑	kalaiarasan	09/18/2014 09:47 AM	09/18/2014 09:50 AM	John Doe	Action
8	☐	[redacted]	☑	John Doe	09/19/2014 01:33 PM	09/19/2014 01:33 PM	John Doe	Action

Step 2: **Click the New Web Service button**
The creation of the web service consists of two sections of information: Web Service Information, and Assignment Rules.



* Required Field(s)

Web Service Information

* Web Service Name:
Active: ✓
* Run Web Service as User: John Doe

Locale Information

IMPORTANT: Locale Information is inherited from the User selected above. To change the Locale for this Web Service, modify the settings in the user record.

Country Locale: United States
Time Zone: GMT

Advanced Administration Information

IMPORTANT: Advanced Administration settings (user permissions) are inherited from the User selected above. To change the Advanced Administration Settings for this Web Service, modify the settings in the user record.

Assignment Rules

Contacts:	[● Specific User] ○ Assignment Rule	John Doe
Inquiries:	[● Specific User] ○ Assignment Rule	John Doe
Leads:	[● Specific User] ○ Assignment Rule	John Doe
Notes:	[● Specific User] ○ Assignment Rule	John Doe
Organizations:	[● Specific User] ○ Assignment Rule	John Doe
Registrations:	[● Specific User] ○ Assignment Rule	John Doe

Step 3: **Create a Web Service Name**
The web service name is an open field and allows for a variety of names. This is a required field, and it is recommended it follow a naming convention that will allow for effective management of the web service accounts.

Step 4: **Run Web Service as User**
Each web service account is required to be assigned to a Radius user. By default, this value is assigned to the user who is creating the account. This relationship determines the system permissions affiliated with the new web service.

<u>Example</u>

In creating a new web service account, the administrator changes the assigned user. This user has limited permissions within Radius, and therefore the web service account will inherit the permission level of the assigned user.

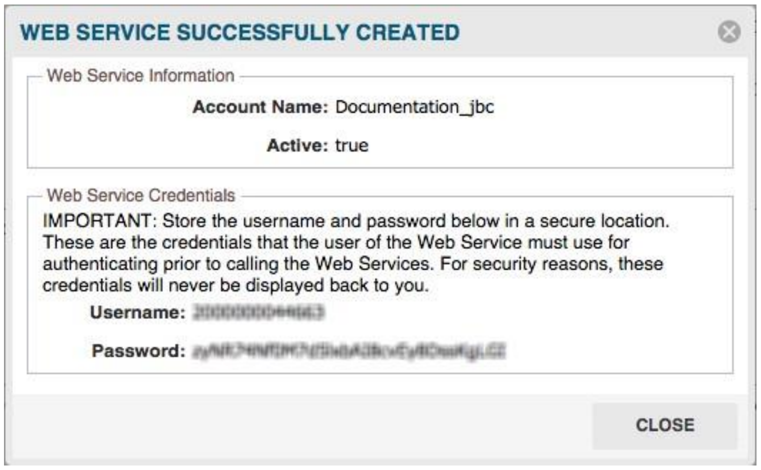
Best Practice: Ensure the permission level of the assigned user account corresponds with the permission level the web service account requires.

Step 5: **Assignment Rules**
By default, the system will set the Assignment Rules to the Radius user who creates the web services account. Similar to the process of creating a contact record, the user can determine whether she owns the record, or the newly created record is routed to another user with Assignment Rules.

Web Service assignment rules determine how new contacts, inquiries, leads, notes, etc, will be routed.

Step 6: Saving the Web Service Account

Click the save button to finish the creation of the account. Upon a successful creation, the user is presented with the following screen.



WEB SERVICE SUCCESSFULLY CREATED

Web Service Information

Account Name: Documentation_jbc

Active: true

Web Service Credentials

IMPORTANT: Store the username and password below in a secure location. These are the credentials that the user of the Web Service must use for authenticating prior to calling the Web Services. For security reasons, these credentials will never be displayed back to you.

Username: [illegible]

Password: [illegible]

Each Web Service account is provided a set of unique credentials that are required to utilize the functionality. As noted in the confirmation message, **it will only be displayed one time.**

We highly recommends that users immediately record these credentials, as well as utilize a secondary method such as a screenshot.

In the event the account credentials are lost, it is possible to generate a new set.

- Navigate to the web services account list
- Click on the name of the web service account
- Click the Edit button at the top of the menu
- Make changes if desired and click the *Save and Generate New Credentials* button
- **Record new credentials and provide to account user**

Once a web services account has been created, the user has the appropriate credentials needed to access Radius. Generating new credentials invalidates the old credentials.

Radius Web Services Use Cases

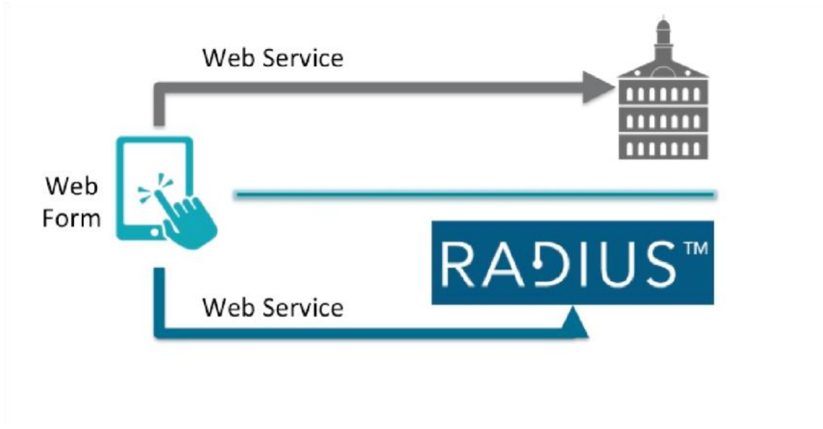
Web services are a versatile tool that
can be used to integrate various systems with Radius. Every Radius client is unique and therefore there are numerous cases where webservices can be utilized.

BEST PRACTICE: If you are uncertain as to how you would use webservices or need assistance in the setup, contact your Campus Management Representative.

Here are some common examples of web services use cases:

Marketing and Recruiting

The University marketing and recruiting office uses a custom form and lead generation site to collect leads. This data must be pushed into Radius, as well as another university system.

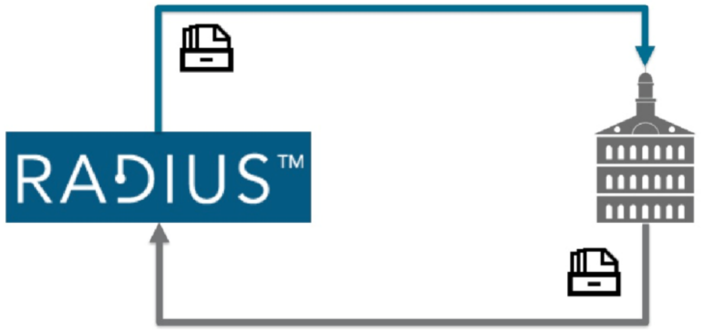


In this scenario the university is utilizing a web form to collect prospect data. Through the web service, data collected is pushed, in real-time, to Radius. The web form can be customized to collect any set of data. The data is also pushed into their SIS in real-time.

To successfully complete this action the user would reference the Create an Entity Web Service.

Real-Time Exchange of Data

The University needs a system that allows for the real-time exchange of data with an existing on-campus system. This exchange is used for contact, interaction, organization, and lifecycle management.



In this scenario the university is utilizing web services to ensure the real-time transfer of data between Radius and the SIS. It is important to note that Web Services should not be used for bulk type transfers. The transfer of bulk data can be achieved via an export/import process.

To successfully complete this action the user would reference the actions for Updating an Entity or Creating an Entity.

In either scenario, users may want to query the system first using the Search for Entities action. This allows users to check for the existence of entities prior to attempting to create new entities.

Available Radius Modules

Radius is built upon numerous modules that provide the user access to a wide set of data points. There are two components to Radius modules and it is important that users note the distinction. Below is a list of the available modules, which represents the "System Names" and not the "Display Names" that appear in the Radius User Interface. Web Services utilize the system names that are returned in the "List All Modules" Web Service. Additional modules will become available in future releases for web services. This list of modules that are returned for your tenant is based on your Radius package and may be less than what is listed below. Check for updates after each release to explore new modules that are available via the Web Services.

```
{
  "status": "ok",
  "payload": [
    "UserNotifications",
    "Accounts", (aka "Organizations")
    "Cases",
    "Programs",
    "Inquiries",
    "Venues", (aka "Locations")
    "RelatedAccounts", (aka "Groups And Associations")
    "Tasks",
    "Notes",
    "Iterations",
    "CaseMessages",
    "Appointments",
    "Registrations",
    "Leads",
    "Contacts",
    "Experiences",
    "Educations",
    "Lifecycles",
    "Recommendations",
    "Connections",
    "Users",
    "Groups",
    "Roles",
    "Profiles",
    "Decisions",
    "Events",
    "Attendees",
    "EventSession",
    "EventSessionAttendees",
    "EventSessionGroup",
    "Segments", (aka "Targets")
    "Requirements",
    "TestScores",
    "ExportFilters",
    "ContactEngagements",
    "ContactEngagementLogs",
    "Campaigns",
    "CommunicationPlans",
    "CommunicationPlansContacts"
  ]
}
```

Available Field Attributes

Modules in Radius are comprised of fields. To ensure the preservation of data, a specific set of field attributes are made available to the web service user.

Attribute	Description
searchFields	An internal system label that is used by web services. This is the label that users should reference in web service calls.

Display Label	The field display label shown in the User Interface. This label should not be referenced in web service calls.
Mandatory	States whether the field is required as part of creating/updating the record.
Custom Field	States whether this is a user-created or a system field.
Decimal Places	For Currency and Number fields, states the number of decimal places as configured in the User Interface.
Maximum Length	States the maximum length of characters allowed for the field value.
Data Type	States the type of the field, as configured in the User Interface.
Groups	For “Grouped Multi-Select” type fields, lists all of the groups with the respective values for each group.
Possible Values	For “Pick List” and “Multi-Select” type fields, lists all of the possible values.
Update Allowed	Indicates data for the field is updateable as part of web service calls.
Create Allowed	Indicates data for the field can be created as part of web service calls.

Web Service Methods for Radius

The examples within this section can be utilized as a reference for accessing data within Radius. For maximum security of data, all Web Service calls (URL's) are required to use HTTPS.

PRODUCTION Environments:

<host> for US clients: api.radiusbycampusmgmt.com
<host> for USEAST clients: useastapi.radiusbycampusmgmt.com
<host> for APAC clients: apacapi.radiusbycampusmgmt.com
<host> for EMEA clients: emeaapi.radiusbycampusmgmt.com

STAGING Environments:

<host> for US clients: api**-staging**.radiusbycampusmgmt.com
<host> for USEAST clients: useast**staging**api. radiusbycampusmgmt.com
<host> for APAC clients: apac**staging**api. radiusbycampusmgmt.com
<host> for EMEA clients: emea**staging**api. radiusbycampusmgmt.com

Please note that all testing of the Radius Web Services should occur in your STAGING environment. This will allow you to work with test data as well as work out the detailed interaction with each Web Service request and response.

The createFields and returnFields parameters accept either the "Field ID" or the "Field Label". We highly recommend referencing the "Field ID" for all Web Service calls.

<u>List All Modules</u>

URL: https://<host>/crm/webservice/modules
HTTP Method: GET
Optional parameters:
useDisplayLabels=true - Return display labels instead of system labels. Default is false.
useSystemAndDisplayLabels=true – Returns system labels and display labels.
Returns: A list of module names available through the web service.

<u>Get a Module's Meta Data</u>

URL: https://<host>/crm/webservice/modules/{moduleName}

HTTP Method: GET
Path variables:
moduleName - a valid module name
Returns: The display label for the module, as configured in the Radius UI as well as a list of the modules fields.

List All Fields for a Module

URL: https://<host>/crm/webservice/modules/{moduleName}/fields
HTTP Method: GET
Path variables:
moduleName - a valid module name
Optional parameters:
includeDetails=true
Returns detailed field information.
Returns the unique "Field ID" per field.
Using the "Field ID" instead of the "Field Label" in the searchFields and returnFields parameters of other Web Service calls guarantees that changing the "Field Label" in the Radius User Interface will not break the Web Services. We highly recommend referencing the "Field ID" for all Web Service calls.

Returns: A list of all the fields available in the module.

Get an Entity

URL: https://<host>/crm/webservice/modules/{moduleName}/{entityId}
HTTP Method: GET
Path variables:
moduleName - a valid module name *entityId* - the unique ID of the entity **Optional parameters**:
returnFields - A comma separated list of fields to include as part of the returnFields URL parameter. Invalid field labels in the returnFields request are ignored.
Returns: The entity with the given ID. If the returnFields parameter is populated, only those fields specified will be returned. Otherwise all available fields will be returned.

Create an Entity

URL: https://<host>/crm/webservice/modules/{moduleName}
HTTP Method: POST
Path variables:
moduleName - a valid module name
Request body: A CreateFields object in JSON format
Returns: The entity ID. If the returnFields parameter is populated, only those fields specified will be returned.
Details:
The CreateFields object consists of:
createFields - A set of field label/field value pairs.
returnFields (optional) - A set of field labels indicating which fields should be returned for the created entity. Defaults to Entity ID. Invalid field labels in the returnFields request are ignored.
Examples:
A sample request body to create a contact:

```
{
    "createFields": {
        "First Name": "aFirstName",
        "Last Name": "aLastName"
    },
    "returnFields": [
        "Entity ID",
        "First Name",
        "Last Name"
    ]
}
```

A sample successful response:

```
{
     "status":"ok",
     "payload":{
           "entity":{
                 "Entity ID":2000000220005,
                 "First Name": "aFirstName",
                 "Last Name": "aLastName"
           }
     }
}
```

Update an Entity

URL: https://<host>/crm/webservice/modules/{moduleName}/{entityId}
HTTP Method: PUT
Path variables:
moduleName - a valid module name

entityId - the unique ID of the entity **Optional parameters**:

returnFields - A comma separated list of fields to include as part of the returnFields URL parameter. Invalid field labels in the returnFields request are ignored.

Request body: JSON containing key/value pairs of field labels and values to update

Returns: The entity ID. If the returnFields parameter is populated, only those fields specified will be returned.

Examples:

A sample request body to update a contact:

```
{
    "createFields": {
        "First Name": "newFirstName",
        "Last Name": "newLastName"
    },
    "returnFields": [
        "Entity ID",
        "First Name",
        "Last Name"
    ]
}
```

A sample successful response:

```
{
     "status":"ok",
     "payload":{
          "entity":{
               "Entity ID":2000000220005
          }
     }
}
```

Delete an Entity

URL: https://<host>/crm/webservice/modules/{moduleName}/{entityId}

HTTP Method: DELETE

Path variables:

moduleName - a valid module name

entityId - the unique ID of the entity

Returns: A message indicating the entity was deleted.

Search for Entities

URL: https://<host>/crm/webservice/modules/{moduleName}/search
HTTP Method: POST
Path variables:
moduleName - a valid module name
Optional parameters:

page - The page to return. Default is 1. Valid range is 1 - pageSize.

pageSize - The number of entities to include in the response. Default is 50. Valid range is 1 - 50.

queryId - The query ID is not required for the initial query, but is required when requesting additional pages (page > 1). **Request body**: A SearchCriteria object in JSON format **Returns**: A list of entities that meet the search criteria.

Details:

The initial version of the Radius web service provides a simple search capability that allows modules to be searched for entities that match a search criteria.

The search criteria consists of

- searchFields - A set of field label/field value pairs. Only entities whose fields are equal to the specified values will be selected. All search terms are ANDed together.
- returnFields - A set of field labels indicating which fields should be returned for entities selected by the search.
- newerThan - Returns records that are equal to or greater than the specified date.
- updatedSince - Returns records that are equal to or greater than the specified date.

If both newerThan and updatedSince are included, they will be ANDed together.

Search results are paginated, with a default page size of 50. A query id is included in the search response, and must be included when requesting additional pages of results.

Examples:

A sample request body to search for contact’s whose first name equals “FIRST” and last name equals “LAST”, and option parameter newThan and updatedSince (in User’s locale and timezone)

```
{
  "searchFields": {
     "First Name": "FIRST",
     "Last Name": "LAST"
  },
  "newerThan": "11/25/2014 06:18 PM",
  "updatedSince": "11/25/2014 06:18 PM",
   "returnFields": [
     "Entity ID",
       "First Name",
       "Last Name"
  ]
}
```

A sample successful response:

```
{
        "status":"ok",
        "payload":{
                "total pages":1,
                "page":1,
                "total entities":2,
                "queryId":"8bb74976-923b-4cd7-ad85-56c7f911e5ed",
                "entities":[ {
                                "Entity ID":2000000134001,
                                "Last Name":"last",
                                "First Name":"first"
                        },
                        {
                                "Entity ID":2000000137003,
                                "Last Name":"LAST",
                                "First Name":"FIRST"
                        }
                ]
        }
}
```

Enhanced Search for Entities

URL: https://<host>/crm/webservice/modules/{moduleName}/enhancedsearch
HTTP Method: POST
Path variables:
moduleName - a valid module name
Optional parameters:
page - The page to return. Default is 1. Valid range is 1 - pageSize.
pageSize - The number of entities to include in the response. Default is 50. Valid range is 1 - 50. *queryId* - The query ID is not required for the initial query, but is required when requesting additional pages (page > 1). **Request**

body: An EnhancedSearch object in JSON format **Returns**: A list of entities that meet the search criteria.

Details:

The enhanced search allows for comparators other than EQUAL. *Checkbox and Grouped Multi-Select fields are not yet supported.* Foreign fields only support the EQUAL operator, and the value must be the Entity ID of the foreign field.

The search criteria consists of

- Filters - A set of Filter objects. Each filter contains a field name, an array of field values, and a comparator. Only entities that meet all the filter criteria will be returned. All search terms are ANDed together. Some field types support filters with a single value, in which case the value array must contain only one value. Other field types support an array of values. IS_EMPTY and IS_NOT_EMPTY support zero values or a single value consisting of the empty string (""). (See details below.)
- returnFields - A set of field labels indicating which fields should be returned for entities selected by the search.

Field labels and Field IDs are supported in filters and returnFields.

Supported Comparators:

- Strings: EQUAL, NOT_EQUAL, CONTAINS, NOT_CONTAINS, STARTS_WITH, ENDS_WITH, IS_EMPTY, IS_NOT_EMPTY.
- Numerics: EQUAL, NOT_EQUAL, LESS_THAN, GREATER_THAN, LESS_EQUAL, GREATER_EQUAL, IS_EMPTY, IS_NOT_EMPTY.
- Pick Lists: same as Strings ○ Only a single search value is possible. Searching on multiple values that are OR'ed together is not possible. Perform multiple searches on single values and combine results to emulate OR type searches for Pick Lists.
- Box Text (Location Tags field): IS_ALL_OF, IS_NOT_ALL_OF, IS_ANY_OF, IS_NOT_ANY_OF, CONTAINS_ALL_OF, DOES_NOT_CONTAIN_ALL_OF, CONTAINS_ANY_OF, DOES_NOT_CONTAIN_ANY_OF, IS_EMPTY, IS_NOT_EMPTY.
- Multi-Select: IS_EMPTY, IS_NOT_EMPTY, CONTAINS, NOT_CONTAINS, IS_ALL_OF, IS_NOT_ALL_OF, IS_ANY_OF, IS_NOT_ANY_OF, CONTAINS_ALL_OF, DOES_NOT_CONTAIN_ALL_OF, CONTAINS_ANY_OF, DOES_NOT_CONTAIN_ANY_OF.
- EntityMulti-Select (Event Hosts & Presenters fields):

CONTAINS_ALL_OF, DOES_NOT_CONTAIN_ALL_OF, CONTAINS_ANY_OF, DOES_NOT_CONTAIN_ANY_OF, IS_EMPTY, IS_NOT_EMPTY.

- PostalCode: EQUAL, NOT_EQUAL, IS_EMPTY, IS_NOT_EMPTY, CONTAINS, DOES_NOT_CONTAIN, STARTS_WITH, ENDS_WITH, LESS_THAN, LESS_EQUAL, GREATER_THAN, GREATER_EQUAL.
- User (Created By, Modified By, & Owner fields): EQUAL, NOT_EQUAL
- Date, DateTime, MonthYear: EQUAL, NOT_EQUAL, LESS_THAN, GREATER_THAN, LESS_EQUAL, GREATER_EQUAL, IS_EMPTY, IS_NOT_EMPTY.

Search results are paginated, with a default page size of 50. A query id is included in the search response and must be included when requesting additional pages of results.

Examples:

A sample request body to search for contacts whose first name equals “first” and last name starts with “la”:

```
{
 "filters": [
          {
          "comparator": "EQUAL",
    "fieldName": "First Name",
    "type": "filter",
    "values": [
          "first"
          ]
          },
          {
          "comparator": "STARTS_WITH",
    "fieldName": "Last Name",
    "type": "filter",
    "values": [
          "la"
          ]
          }
 ],
 "returnFields": [
          "First Name",
          "Last Name",
          "Entity ID"
 ]
}
```

A sample successful response:

```
{
	"status":"ok",
	"payload":{
		"total pages":1,
		"page":1,
		"total entities":2,
		"queryId":"8bb74976-923b-4cd7-ad85-56c7f911e5ed",
		"entities":[
			{
				"Entity ID":2000000134001,
				"Last Name":"last",
				"First Name":"first"
			},
			{
				"Entity ID":2000000137003,
				"Last Name":"lastTwo",
				"First Name":"first"
			}
		]
	}
}
```

A sample request body containing multiple items in the filter value array:

```
{
 "filters": [
  {
   "comparator": "CONTAINS_ANY_OF",
   "fieldName": "Tags",
   "type": "filter",
   "values": [
    "Tag1", "BadTagName"
   ]
  }
 ],
 "returnFields": [
  "Location",
  "Tags"
 ]
}

The response:
{
 "status": "ok",
 "payload": {
   "entities": [
     {
       "Tags": "Tag1,Tag2",
       "Location": "Test Location"
     }
   ],
   "total pages": 1,
   "page": 1,
   "total entities": 1,
   "queryId": "2e71e943-2d30-4250-a91d-9207c9609ae5"
 }
}
```

Execute a Target

URL: https://<host>/crm/webservice/modules/Segments/execute/<entityid>
HTTP Method: POST
Path variables: *entityId* - the unique ID of
the entity **Optional parameters**:

page - The page to return. Default is 1. Valid range is 1 - pageSize.
pageSize - The number of entities to include in the response. Default is 50. Valid range is 1 - 50. *queryId* - The query ID is not required for the initial query, but is required when requesting additional pages (page > 1).

Returns:

A list of entities that meet the criteria of the target. (Note that some exclusions are automatically applied. See “Module Specific Details” below.)
Targets return Contact data only, so you may specify any field from the contact module to be returned.
Target Preview in the product does not exclude contacts that are in the exclusions, so it shows all contacts based on the Target criteria. Based on the criteria of the target, the Preview in the product may include duplicate contact records. For instance, if the target criteria returns multiple education records for a single contact, that contact will appear in the Preview results multiple times.

In the Target Execute for Web Services, only unique contacts are returned, which is why you may sometimes see less than 50 contacts in the paged search results.

Exclusion Preview in the product works on the exclusion criteria, and includes both system and custom exclusions.

Details:

The execute criteria consists of:

returnFields - A set of field labels (from the Contacts module) indicating which fields should be returned for the entities.

Search results are paginated, with a default page size of 50. A query id is included in the search response, and must be included when requesting additional pages of results. **Example**:

```
{
   "returnFields": [
     "Entity ID",
      "First Name",
      "Last Name"
   ]
}
```

Export Filters – CreateExecutionTask

URL: https://<host>/crm/webservice/modules/ExportFilters/createExecutionTask/<exportFilterId>

HTTP Method: POST

Path variables: *exportFilterId* - the unique ID of the Export Filter **Optional parameters**:

None.

Returns:

On success, returns an “Execution Task ID” that can be used in the GetExecutionTask Web Service.

On failure, returns an appropriate error message.

Examples:

```
{
  "status": "ok",
  "payload": {
    "Execution Task ID": "2000000160005"
  }
}

{
  "status": "error",
  "payload": {
```

```
  "Error Message": "Entity ID specified is invalid."
 }
}

{
 "status": "error",
 "payload": {
  "Error Message": "Concurrent export filter executions are not allowed."
 }
}

{
 "status": "error",
 "payload": {
  "Error Message": "The Export Filters: Max Allowed Executions Per
Day of 5 was reached. Contact your Account Manager for more
information."
 }
}

{
 "status": "error",
 "payload": {
  "Error Message": "The Export Filters: Max Allowed Simultaneous
Executions of 3 was reached. Contact your Account Manager for more
information."
 }
}
```

Export Filters – GetExecutionTask

URL: https://<host>/crm/webservice/modules/ExportFilters/getExecutionTask/<executionTaskId>
HTTP Method: GET
Path variables:

executionTaskId - the executionTaskID returned from the CreateExecutionTask Web Service.

Optional parameters:

returnFileContents=true – Returns the file contents as part of the response. Note that the file contents are encoded and zipped. You must first "BASE64 Decode" the file contents, then unzip the file contents to get the comma-delimited data in its raw text format.
If you wish to not return the File Contents are part of executing the GetExecutionTask Web Service, you must use the

GetExecutionTaskResult Web Service to return results in a paginated fashion. This works similar to the search Web Services.

Returns:

"Execution Task Status" is returned as "Finished" if the Export Filter ran successfully and exported the results.

If "Execution Task Status" is not returned as "Finished", you'll need to call the GetExecutionTask Web Service until it returns the "Finished" status.

To return the records as part of a Web Service Response in a paginated fashion, use the GetExecutionTaskResult Web Service.

On failure, returns an appropriate error message. **Example**:

```
{
   "status": "ok",
   "payload": {
      "Status Date Time": "2017-11-29 20:02:40.0",
      "Export Filter ID": "849000007143047",
      "Created Date Time": "2017-11-29 20:02:40.0",
      "Execution Task Status": "Finished",
      "Execution Task ID": "849000007142159",
      "Total Records": 5
   }
}
```

```
{
  "status": "ok",
  "payload": {
    "Status Date Time": "2017-11-03 17:09:17.0",
    "Export Filter ID": "849000007129027",
"Created Date Time": "2017-11-03 17:09:17.0",
"File Content":
"UEsDBBQACAAAAAAAgIAHRSY0sAAAAAAAAAAAAAAAAhAA
AAYWxsY29udGFjdHNORVdfMjAxNzExMDMxMDA5\nMTcuY3N2j
VXLbtswELwX6D8YOheCRD3s3IQ82qZt0gIJULQ3SmUs1hRpS1QN59
d6yCf1F8qhbTh6\nMfHB1NAc784sd/Xv75N3oaSmhZ5dX3rvvPe8bvTsl
lbMgC/09HxVUS68t2+8RXwW4DMPCQnOFuYn\nzRp9WvpnogBnzlW
e78x6v+UyN6uSzNfmmWYMf+wXqhoQwwBELgSId7yqlGzM09bscFpl
y2kiMccu\nOZXsELEwK10yqcNMKtk246zYnPqqS1Yf5RJkKpVW2LR
5Ele66YBPoRQ7Th78uVVHUjjmIbFWMK13\ntixWWA7oCwOypiiVEj
771Q6ZxHrAxB9wruWm5TX+ZE1r44cjLQI7PrGHBxtZCFj/28AsBxinw
IGL\nsi1W4FC5AqcA9nOgrFR6Oh5suCt52UIiN18NgC+4S14EY27Ykk
qw9ewb06xG2Ap7jV5b7JAZwZ8P\nXEqYcsP3OpfAfgXkDB5bvbQWr
OkotjsHzdPXNEo7hSdjhY/gypXgj9SUuzxcZ/qsgRzS4mBKWmal\njZOI
Lfue9LO1RYcZj2222UyHsr0ju1fMpJjxPRondfVHY/pj6P8+mAG52fGb
Pcy2ql6NBkjcXTNd\nmYT0WzmidhScJkHkmlzJYJREeZ+fu/hw5p5W1
e5ZxfVWvVTxBG595nL2g886NVxxueO2kK4qpkE/\n7TA/TjBH1JR0en
DdaUH/FT2Yxp2LkIxdhDQ9vAzG3gUvX4V0MZBG+8PdkeEcznxkgsnj
cCoBMJwc\nfs5JRxdUev8BUEsHCCufF7k9AgAAcgcAAA==",
   "Execution Task Status": "Finished",
   "Execution Task ID": "849000007129030"
 }
}

{
 "status": "error",
 "payload": {
   "Error Message": "Execution Task ID specified is invalid.",
   "Execution Task Status": "Error"
 }
}

{
 "status": "error",
 "payload": {
   "Error Message": "There are no fields defined for the export filter,
please select fields from edit fields layout and continue.",
   "Execution Task Status": "Error"
 }
}
```

Export Filters – GetExecutionTaskResult

URL: https://<host>/crm/webservice/modules/ExportFilters/getExecutionTaskResult/<executionTaskId>
HTTP Method: GET
Path variables:
executionTaskId - the executionTaskID returned from the CreateExecutionTask Web Service.
Optional parameters:
page - The page to return. Default is 1. Valid range is 1 - pageSize.
pageSize - The number of entities to include in the response. Default is 50. Valid range is 1 - 50. *queryId* - The query ID is not required for the initial query, but is required when requesting additional pages (page > 1).
Returns:
A list of entities that meet the criteria of the Export Filter.
Details:

Search results are paginated, with a default page size of 50. A query id is included in the search response, and must be included when requesting additional pages of results. **Example**:

```
{
   "status": "ok",
   "payload": {
      "entities": [
         {
            "Email": "test1@campusmgmt.com",
            "First Name": "First1",
            "Gender": "Female",
            "Contact ID": "849000007137015",
            "Last Name": "Last1"
         },
         {
            "Email": "test2@campusmgmt.com",
            "First Name": "First2",
            "Gender": "Male",
            "Contact ID": "849000007137017",
            "Last Name": "Last2"
         },
{
            "Email": "test3@campusmgmt.com",
            "First Name": "First3",
            "Gender": "Male",
            "Contact ID": "849000007137019",
            "Last Name": "Last3"
         },
         {
            "Email": "test4@campusmgmt.com",
            "First Name": "First4",
            "Gender": "Male",
            "Contact ID": "849000007137117",
            "Last Name": "Last4"
         },
         {
            "Email": "test5@campusmgmt.com",
            "First Name": "First5",
            "Gender": "Male",
            "Contact ID": "849000007137119",
            "Last Name": "Last5"
         }
      ],
      "total pages": 1,
      "page": 1,
      "total entities": 5,
      "queryId": "a741f2b1-4ae9-46a6-b72a-97177baf2708"
   }
}
```

When no records are returned:

```
{
  "status": "ok",
  "payload": {
    "entities": [],
    "total pages": 0,
    "page": 0,
    "total entities": 0,
    "queryId": "5e619bff-1d5b-4542-9908-94337d0833f9"
  }
}
```

Test Scores - GetTestScoreTypeComponents

URL: https://<host>/crm/webservice/modules/ TestScores/getTestScoreTypeComponents?testScoreType=<testScoreType>
HTTP Method: GET
Optional parameters:
testScoreType – Return only TestScoreTypeComponents for the testScoreType specified.
Returns:
A list of all TestScoreTypeComponents for all TestScoreTypes. **Example**:

```
{
    "status": "ok",
    "payload": {
        "ACT": [
            {
                "Max Score": 36,
                "Min Score": 0,
                "Name": "English",
                "Mandatory": true
            },
            {
                "Max Score": 36,
                "Min Score": 0,
                "Name": "Math",
                "Mandatory": true
            },
        ],
        "PSAT": [
            {
                "Max Score": 80,
                "Min Score": 20,
                "Name": "Math",
                "Mandatory": true
            },
            {
                "Max Score": 80,
                "Min Score": 20,
                "Name": "Critical Reading",
                "Mandatory": true
            },
            {
                "Max Score": 80,
                "Min Score": 20,
                "Name": "Writing",
                "Mandatory": true
            }
        ]
    }
}
```

Create a Custom Field

URL: https://<host>/crm/webservice/modules/{moduleName}/createField
HTTP Method: POST
Path variables:
moduleName - a valid module name
Request body: A CustomFieldData object in JSON format **Optional parameters**: *returnFields* - A comma separated list of fields to include as part of the returnFields URL parameter.
showTypes - if set to true, will just return UIType String name and Integer value pair, since this is a POST, an empty JSON value is required for the request body. A custom field will not be created.

Returns: The field ID. If the field cannot be created, a message indicating the problem will be included in the response.
Details:
The CustomFieldData object consists of

- fieldInfo - A set of field label/field value pairs.

Examples:

A sample list of request body to create a contact field:

For “Text Field” field

```
{
  "fieldInfo": {
        "Type": "Text Field",
        "Field Label": "My Text Field",
        "Length": 3
  }
}
```

For “Integer” field

```
{
  "fieldInfo": {
        "Type": "Integer",
        "Field Label": "My Integer Field",
        "Length": 4
  }
}
```

For “Percent” field

```
{
  "fieldInfo": {
        "Type": "Percent",
        "Field Label": "My Percent Field"
  }
}
```

For “Currency” field

```
{
  "fieldInfo": {
        "Type": "Currency",
        "Field Label": "My Currency Field",
        "Length": 2
  }
}
```

For “Date” field

```
{
  "fieldInfo": {
        "Type": "Date",
          "Field Label": "My Date Field"
  }
}
```

For “Email” field

```
{
  "fieldInfo": {
          "Type": "Email",
          "Field Label": "My Email Field"
  }
}
```

For “Phone” field

```
{
  "fieldInfo": {
          "Type": "Phone",
          "Field Label": "My Phone Field",
          "Length": 10
  }
}
```

For “Pick List” field

```
{
  "fieldInfo": {
          "Type": "Pick List",
          "Field Label": "My Pick List Field",
          "sortValues": true,
          "firstIsDefault": true,
          "values":[
 {"value":"a","description":"","public":true,"systemDefined":"","ca
nChangeVisibility":true,"color":""},   {"value":"b","description":"","publi
c":true,"systemDefined":"","canChangeVisibility":true,"color":""},   {"val
ue":"c","description":"","public":true,"systemDefined":"","canChangeVisi
bility":true,"color":""}
          ]
  }
}
```

For “URL” field

```
{
  "fieldInfo": {
          "Type": "URL",
          "Field Label": "My URL Field"
  }
}
```

For “Text Area” field

```
{
  "fieldInfo": {
        "Type": "Text Area",
        "Field Label": "My Text Area Field"
  }
}
```

For “Checkbox” field

```
{
  "fieldInfo": {
        "Type": "Checkbox",
        "Field Label": "My Checkbox Field",
        "enableByDefault": true
  }
}
```

For “Multi-Select” field

```
{
  "fieldInfo": {
        "Type": "Multi-Select",
        "Field Label": "My Multi-Select Field",
        "sortValues": true,
        "values":[
        {"value":"a","description":"","public":true,"systemDefined":"","ca
nChangeVisibility":true,"color":""},    {"value":"b","description":"","publ
ic":true,"systemDefined":"","canChangeVisibility":true,"color":""},    {"v
alue":"c","description":"","public":true,"systemDefined":"","canChangeVi
sibility":true,"color":""}
        ]
  }
}
```

For “Date/Time” field

```
{
  "fieldInfo": {
        "Type": "Date/Time",
        "Field Label": "My Date/Time Field"
  }
}
```

For “Month/Year” field

```
{
  "fieldInfo": {
            "Type": "Month/Year",
            "Field Label": "My Month/Year Field"
  }
}
```

For “Number” field

```
{
  "fieldInfo": {
            "Type": "Number",
            "Field Label": "My Number Field",
            "decimalPlaces": 3
  }
}
```

For “Auto-Number” field

```
{
  "fieldInfo": {
            "Type": "Auto-Number",
            "Field Label": "My Auto-Number Field",
            "prefix": "pre",
            "suffix": "suf",
            "startingNumber": 2,
            "existingRecords": false
  }
}
```

For “Grouped Multi-Select” field

```
{
  "fieldInfo": {
            "Type": "Grouped Multi-Select",
            "Field Label": "My GroupMultiSelect Field",
            "groups":[

            {"groupName":"a","values":[         {"value":"a","description":"","p
ublic":true,"systemDefined":"","canChangeVisibility":true,"color":""},
{"value":"b","description":"","public":true,"systemDefined":"","canCha
ngeVisibility":true,"color":""}]},
          {"groupName":"c","values":[          {"value":"d","description":"","
public":true,"systemDefined":"","canChangeVisibility":true,"color":""},
{"value":"e","description":"","public":true,"systemDefined":"","canCh
angeVisibility":true,"color":""}]}
          ]
  }
}
```

For “Postal Code” field

```
{
  "fieldInfo": {
          "Type": "Postal Code",
          "Field Label": "My Postal Code Field",
          "Length": 10
  }
}
```

A sample successful response:

```
{
          "status":"ok",
          "payload":{
                    "field":{
                              "Field  ID":2000000063001
                    }
          }
}
```

<u>Campaigns – getStatistics</u>

URL: https://<host>/crm/webservice/modules/Campaigns/getStatistics/{entityId}
HTTP Method: GET
Path variables:
entityId - the unique ID of the Campaign
Optional parameters:
None.
Returns:
On success, returns campaign statistics.
On failure, returns an appropriate error message.

Examples:
A Sample successful response:

```
{
   "status": "ok",
   "payload": {
      "Campaign Status": "Completed",
      "Email Address To Send To": "Default Email Address",
      "Emails Not Viewed": 0,
      "Total Bounce": 1,
      "Hard Bounce": 0,
      "Campaign ID": 83000000972088,
      "Sent By": "Campaign User",
      "Opted Out": 0,
      "Total Targets": 1,
      "Sender Email": "campaignuser@campusmgmt.com",
      "Targets Skipped": 1,
      "Completed Date/Time": "21/02/2021 12:01:04 PM",
      "Emails Not Sent": 1,
      "Start Date/Time": "21/02/2021 12:01:00 PM",
      "URL Not Visits": 0,
      "Campaign Type": "Campaign",
      "Emails Viewed": 0,
      "Subject": "Campaign-Feb 21 - 1",
      "Scheduled Date/Time": "21/02/2021 12:00:00 PM",
      "URL Visits": 0,
      "URL Visits Info": {},
      "Emails Not Sent Due to Engagement Filter": 0,
      "Emails Sent": 0,
      "Soft Bounce": 1,
      "Created By": "Campaign User ",
      "Cases": 0,
      "Campaign Name": "Campaign-Feb 21 - 1"
   }
}
```

Sample Error Response:

```
{
   "status": "error",
   "payload": {
      "Error Message": "CampaignId specified is invalid."
   }
}
```

Communication Plans – getStatistics

URL: https://<host>/crm/webservice/CommunicationPlans/getStatistics/{entityId}
HTTP Method: GET
Path variables:
entityId - the unique ID of the Communication Plan
Optional parameters:
None.
Returns:
On success, returns communication Plan statistics.
On failure, returns an appropriate error message.

Examples:
A Sample successful response:

```
{
   "status": "ok",
   "payload": {
      "Communication Plan ID": 83000002594336,
      "Communication Plan Statistics": [
         {
            "Email Template": "Email Template:: with images  link:
Cloned",
            "Visited URL": 1,
            "Emails Bounced": 23,
            "Days": 0,
            "URL Visits Info": {
               "https://www.google.com": 2
            },
            "Emails Sent": 64,
            "Emails Viewed": 27,
            "Opted Out": 0
         },
         {
            "Email Template": "Email template:: Complan Day 7: Sendgrid:
New",
            "Visited URL": 0,
            "Emails Bounced": 25,
            "Days": 1,
            "URL Visits Info": {},
            "Emails Sent": 65,
            "Emails Viewed": 18,
            "Opted Out": 0
         },
         {
            "Email Template": "",
            "Visited URL": 1,
            "Emails Bounced": 48,
            "Days": "Total",
            "URL Visits Info": {
               "https://www.google.com": 2
            },
            "Emails Sent": 129,
            "Emails Viewed": 45,
            "Opted Out": 0
         }
      ],
      "Communication Plan Name": "New Complan : to check statistics"
   }
}
```

Sample Error Responses:

```
{
    "status": "error",
    "payload": {
        "Error Message": "Communication Plan ID specified is invalid."
    }
}
```

Print Registrations – CreatePrintRegistrationExecutionTask

URL: **Error! Hyperlink reference not valid.**
HTTP Method: POST
Optional parameters:
None
Request Body: Registration Fields,
registrationIds (user can pass up to 100 registration Ids.)
and skipInvalidRegistrationIds
Details:
printOptions- A list of selected print options
registrationIds - A set of registration Ids.
skipInvalidRegistrationIds - can pass true or false, if skipInvalidRegistrationIds = true, it will skip the invalid registration Ids.

Sample request body:
```
{
  "printOptions": [
    "APPLICATION_FORM",
    "REGISTRATION_NOTES",
    "REGISTRATION_DETAILS"
  ],
  "registrationIds":[
    "2000000047057","2000000047068",
    "2000000047079"
],
  "skipInvalidRegistrationIds": true
}
```

Returns:

On success, returns an “Execution Task ID” that can be used in the GetPrintRegistrationExecutionTask and GetPrintRegistrationExecutionTaskResult Web Service.

On failure, returns an appropriate error message.

Examples:
```
{
  "status": "ok",
  "payload": {
    "Execution Task ID": "2000000160005"
  }
}

{
  "status": "error",
  "payload": {  "Error Message": "RegistrationIds specified are invalid.
["2000000047077"]"
  }
}
{
    "status": "error",
    "payload": {
        "Error Message": "Execution Task ID specified is
      invalid :[2000000046018]"
    }
}


{
  "status": "error",
  "payload": {
    "Error Message": "Webservices: Max PDF Print Allowed Simultaneous
Requests of 5 was reached. Contact your Account Manager for more
information."
  }
}

{
  "status": "error",
  "payload": {
    "Error Message": "Webservices: Max PDF Print Allowed Simultaneous
Requests of 100 was reached. Contact your Account Manager for more
information."
  }
}
```

Print Registrations – GetPrintRegistrationExecutionTask

URL: **Error! Hyperlink reference not valid.**

HTTP Method: GET
Path variables:
executionTaskId - the executionTaskID returned from the **CreatePrintRegistrationExecutionTask** Web Service.
Optional parameters:
returnFileContents=true – Returns the BASE64 encoded file contents as part of the response. Note that the file contents are encoded.You must first "BASE64 Decode" the file contents.
Returns:
"Execution Task Status" is returned as "Finished" if the BASE64 encoded file contents store in database successfully. If "Execution Task Status" is not returned as "Finished", you'll need to call the GetPrintRegistrationExecutionTask Web Service until it returns the "Finished" status.
Example:

```
{
    "status": "ok",
    "payload": {
        "Created Date Time": "2022-09-28 10:02:29.0",

        "Execution Task Status": "Pending",

        "Execution Task ID": "83000005212001"
    }
}

{
    "status": "ok",
    "payload": {
        "Created Date Time": "2022-09-28 10:02:29.0",
        "File Content":
"JVBERi0xLjQKJeLjz9MKMiAwIG9iago8PC9GaWx0ZXIvRmxhdGVE
ZWNvZGUvTGVuZ3RoIDY2Nz4+c3RyZWFtCnictVZdb9MwFLXUN7/
xD/zIQPJsJ/7IK+q6ahS20owiKC9oGtO0IG0v/H2OP5LFSRMkJlSlucc+9
+Tc69tsj/RdTQVzpeTOsfqGntV0Sx+p4KKwmv2mil1g/55KwT7Qb98Fu
6GFYU5oz2+o1hVXrsUPLS4MFyUgqL2wY93RPf2FZzjWXk8/aamYdZ
IXMsgWXiZhL6u4LoAF1wa4VBzuOnhHb99k+YP9BJXipggQsWytILW
Vj4Rmjt86G+T37TYju7n+hIhk/hMa0TM/LYZOg9YVN4C91owb/Rd9y
S3M2Yo73/uy4PoZpVZnCjkjIoO9sbnUrfiAQGlm+cNas9N6LiA3nOlPa
OBIjSv9QjdsCXcJwJ2hGI8y+5spFgWvIgyxrAauE6GZ47eGBvl9l03uMh
efUBgMmM+ckDEpLREHsO3F8bk6KhsOxDiY0u2EdKg3UTE3347IdG
3KHPXnKXGa+YxhidnZJOu521x+QkEw/0ET8Do9XaHPfvbqW/Tcr0u
WXBSWC6w39DX5QSx5TzbkM1mQHVmSC7JG9BUrl/gsyHnY2yFeks
MBC0uyjcE1mDX5gn0fXeHakn3Q2UNxie8FVlPWWZDNU86Dmk8A5
ROWF2SF+1V42hr566S1DsRdWN+A47U3YG7Jq5P63v+tmC9XWI65j
uWmBMGNFbrCzAgprcFdKVvqudYZ/34oSyRGqcMTbJ2GCnypEhGa0
Oq3BxHTlOVW4VCrkhcdfOig4SbMSOTmKFH9oM86U3jtmOjsLaxcor
3+qD7i+/oFRTs/bdY5rnWvbIZLjw7zcAKw9o8ediHIq7G8Dq9ZY7gqo7
oLQ7TBdZlEDe6r9sBfXIhf/4dC5lVNJXlV/a/2GFN1M6eP/+aSnv+PaUv/
ADlUvSkKZW5kc3RyZWFtCmVuZG9iago2IDAgb2JqCjw8L0xlbmd0a
DEgMzc2NjAwL0ZpbHRlci9GbGF0ZURlY29kZS9MZW5ndGggNjUzN
Tk+PnN0cmVhbQp4nOy9C3yVxbU3vOa57uzcdm7shBBJjDFgihFiCBE
wOVxCBIwUMcZIQqQBYhRCihgQaaBpGgERKAVp5FBExDSlNCJFp
Ei9IaJGq5YiteqxSNUe8LWWIsVk7/e/5pmd7ISg9u15v3O+3/dl5581lzUz
a9asmVmzn0tIEFEILSOdPN+5+67k0VP7FyBlM1HYotnz58z9+dXzthO
FxxLZ1XPuXDy...

 "Execution Task Status": "Finished",
 "Execution Task ID": "83000005212001"
    }
}

{
  "status": "error",
  "payload": {
    "Error Message": "Execution Task ID specified is invalid.",
    "Execution Task Status": "Error"
  }
}
```

Print Registrations– GetPrintRegistrationExecutionTaskResult

URL: **Error! Hyperlink reference not valid.**

HTTP Method: GET
Path variables:
executionTaskId - the executionTaskID returned from the **CreatePrintRegistrationExecutionTask** Web Service.
Optional parameters:
none
Returns:
Returns the BASE64 encoded file contents as part of the response.
Example:

```
{
  "status": "ok",
  "payload": {
    "Created Date Time": "2022-09-28 10:02:29.0",
    "File Content":
"JVBERi0xLjQKJeLjz9MKMiAwIG9iago8PC9GaWx0ZXIvRmxhdGVE
ZWNvZGUvTGVuZ3RoIDY2Nz4+c3RyZWFtCnictVZdb9MwFLXUN7/
xD/zIQPJsJ/7IK+q6ahS20owiKC9oGtO0IG0v/H2OP5LFSRMkJlSlucc+9
+Tc69tsj/RdTQVzpeTOsfqGntV0Sx+p4KKwmv2mil1g/55KwT7Qb98Fu
6GFYU5oz2+o1hVXrsUPLS4MFyUgqL2wY93RPf2FZzjWXk8/aamYdZ
IXMsgWXiZhL6u4LoAF1wa4VBzuOnhHb99k+YP9BJXipggQsWytILW
Vj4Rmjt86G+T37TYju7n+hIhk/hMa0TM/LYZOg9YVN4C91owb/Rd9y
S3M2Yo73/uy4PoZpVZnCjkjIoO9sbnUrfiAQGlm+cNas9N6LiA3nOlPa
OBIjSv9QjdsCXcJwJ2hGI8y+5spFgWvIgyxrAauE6GZ47eGBvl9l03uMh
efUBgMmM+ckDEpLREHsO3F8bk6KhsOxDiY0u2EdKg3UTE3347IdG
3KHPXnKXGa+YxhidnZJOu521x+QkEw/0ET8Do9XaHPfvbqW/Tcr0u
WXBSWC6w39DX5QSx5TzbkM1mQHVmSC7JG9BUrl/gsyHnY2yFeks
MBC0uyjcE1mDX5gn0fXeHakn3Q2UNxie8FVlPWWZDNU86Dmk8A5
ROWF2SF+1V42hr566S1DsRdWN+A47U3YG7Jq5P63v+tmC9XWI65j
uWmBMGNFbrCzAgprcFdKVvqudYZ/34oSyRGqcMTbJ2GCnypEhGa0
Oq3BxHTlOVW4VCrkhcdfOig4SbMSOTmKFH9oM86U3jtmOjsLaxcor
3+qD7i+/oFRTs/bdY5rnWvbIZLjw7zcAKw9o8ediHIq7G8Dq9ZY7gqo7
oLQ7TBdZlEDe6r9sBfXIhf/4dC5lVNJXlV/a/2GFN1M6eP/+aSnv+PaUv/
ADlUvSkKZW5kc3RyZWFtCmVuZG9iago2IDAgb2JqCjw8L0xlbmd0a
```

DEgMzc2NjAwL0ZpbHRlci9GbGF0ZURlY29kZS9MZW5ndGggNjUzN
Tk+PnN0cmVhbQp4nOy9C3yVxbU3vOa57uzcdm7shBBJjDFgihFiCBE
wOVxCBIwUMcZIQqQBYhRCihgQaaBpGgERKAVp5FBExDSlNCJFp
Ei9IaJGq5YiteqxSNUe8LWWIsVk7/e/5pmd7ISg9u15v3O+3/dl5581lzUz
a9asmVmzn0tIEFEILSOdPN+5+67k0VP7FyBlM1HYotnz58z9+dXzthO
FxxLZ1XPuXDy...

```
 "Execution Task Status": "Finished",
 "Execution Task ID": "83000005212001"
    }
}

{
  "status": "error",
  "payload": {
    "Error Message": "Execution Task ID specified is invalid.",
    "Execution Task Status": "Error"
  }
}

{
    "status": "error",
    "payload": {
        "Error Message": "Print Registration execution is not yet finished or
                  has an error. Please use getPrintRegistrationExecutionTask
                  endpoint to get the status of an execution task."
    }
}
```

Module Specific Details

Users

- The Users web service accepts a mail profile ID or creates a default mail profile based on the user's name and email fields.
- When searching for users, the “Role” field accepts the string value of the “Role Name”. (Searching by the string value is a bug and will be changed to searching by “Entity ID” in a future release.)
- Sample JSON for searching for users by role:

```
{
  "searchFields": {
      "Role": "President"
  },
   "returnFields": [
     "Entity ID",
      "First Name",
      "Last Name"
  ]
}
```

- When creating users, the “Role” field accepts the “Entity ID” value of an existing role. Use the “search” web service to return the “Entity ID” of the role, based on the “Role Name” field from the “Roles” module.
- When creating users, the “Profile” field accepts the “Entity ID” value of an existing profile. Use the “search” web service to return the “Entity ID” of the profile, based on the “Profile Name” from the respective module.
- Sample JSON for creating a new user:

```
{
   "createFields": {
      "First Name": "Web Services First Name",
      "Last Name": "Web Services Last Name",
      "Email": "webservices@webservice.com",
      "Role": "165000000310239",
      "Profile": "165000000310245",
      "Language Locale": "English",
      "Country Locale": "United States",
      "Time Zone": "( GMT -5:0 ) Eastern Standard Time
(Canada/Eastern)"
   },
   "returnFields": [
      "Entity ID",
      "First Name",
      "Last Name"
   ]
}
```

Profiles

- Profile permission management is not available through web services. A Profile can be created, and like the User Interface a "Cloned Profile" is required - permissions are based on the Cloned Profile.

- Deleting a Profile through web services will transfer all associated users to the Standard Profile. In the Radius User Interface, the "transfer to" Profile is a required parameter when deleting a Profile. Adding a "transfer to" option for

Profile delete breaks the paradigm of the web service calls, so we have defaulted to the Standard Profile.

Roles

- Deleting a Role will move all the Role's users to the Role's "Reports To" Role. If the Role does not have a "Reports To" Role, then it cannot be deleted using web services.

Inquiries

- When creating a new inquiry, add "Type" : "Inquiry" to the CreateFields object. This will properly scope the Case being created as an Inquiry, since an Inquiry is a type of Case in Radius.

CaseMessages

- Expected format for “Sent Date” field: “MM/DD/YYYY HH:mm AM/PM”
- Example: “03/10/2012 12:21 AM”

Registrations

- Must include the “Participant” and “Iteration Name” when updating existing Registrations.

Events

- The following fields in the Events have a discrepancy between the “System Label” and the “Display Label” that is seen in the Radius User Interface
  ○ Display Label = System Label
  ○ Waitlist Message = Overflow Registration Message ○
  Allow Waitlist = Allow Overflow Registrations
- Recurring Events can be read via the Web Services, but cannot currently be modified.

Multi-Select Field Format

- In order to set the values for Multi-select fields, you must use the following format:
- Example “createFields” section of body:

```
"createFields": {
   "Mutli-Select Field Name": ["Value1","Value2","Value3"]
},
```

Targets

- Targets are exposed to the Web Services as “Segments”.
- Create/Update/Delete of Targets are not supported via Web Services.
- How Exclusions Apply to Targets

- Campaign Publication Types and Contact Preferences DO NOT come into play when executing Targets via the Web Services, unless you build criteria into your Targets based on the Contact Preferences field on the Contacts module.
- Contacts that are part of any custom (user-created) exclusions in the system WILL NOT be returned when executing Targets via the Web Services.
- Contacts that are part of any system exclusions (No General Mail, etc.) WILL be returned, as system exclusions DO NOT apply when executing Targets via the Web Services.

Requirements

- When creating/updating Requirements via the Web Services, the Requirement Details are not required.
- For "Official Transcript" and "Unofficial Transcript" requirements, the "Education" Requirement Detail is not required.
- For "Recommender" requirements, the "Recommendation" Requirement Detail is not required.
- For "Custom Field" requirements, the "Custom Field" Requirement Detail is not required.
- For "Test Score" requirements, the "Test Score" Requirement Detail is not required.
- The reason for not requiring the "Requirement Details" as part of the Web Services is because client use cases dictate that the requirement be created, but not yet fulfilled (This is similar to how requirements are configured for Iterations). The requirement will be fulfilled later via the standard requirement-fulfilling user interface methods within Radius.

Export Filters

- The standard Web Services can be used to return the module details for Export Filters, and to search for Export Filters. Otherwise, use the 3 new Web Services to execute Export Filters and return results:
- CreateExecutionTask
- GetExecutionTask
- GetExecutionTaskResult

Test Scores

- The standard Web Services can be used to return the module details for Test Scores, and to search, create, delete, and create fields for Test Scores. In addition, a new Web Service to get the TestScoreTypeComponents is available.
- TestScores/search does not support searching on “Test Date” or “Test Score Values”. Use TestScores/enhancedsearch instead.
- TestScores/enhancedsearch – Supports enhanced searching, including searching on Test Type and Test Type Component. Sample JSON for enhanced searching: (Note the new "testScoreTypeComponentFilter" type below.)

```
{
  "filters": [
  {
    "comparator": "GREATER_THAN",
    "testType": "ACT",
    "testScoreTypeComponent": "Math",
    "type": "testScoreTypeComponentFilter",
    "values": [
      20
    ]
  }
  ],
  "returnFields": [
    "Test Score Values",
    "Test Type",
    "Test Date",
    "Contact",
    "Entity ID"
  ]
}
```

- TestScores/enhancedsearch limitations:
- Normal enhancedsearch (via simple "filters") can be performed for all fields of Test Score, with the exception of Test Score Values. Use the new "testScoreTypeComponentFilter" type from the example above.
- A maximum of 2 "testScoreTypeComponentFilter" comparisons is allowed in a single search request.
- Searching across multiple "testScoreTypeComponentFilter" Types in a single request is not allowed.
- Test Scores – Create an Entity
- Pass in fields of Test Score Values as JSON data in createFields
- "Test Score Values": {"Critical Reading":300.0,"Writing":200.0,"Math":600.0}
- Test Scores – Update an Entity
- When updating "Test Score Values", all mandatory Test Score Components must be passed in, even if you are only updating values for some components.

Campaigns

- The standard Web Services can be used to return the module details for campaigns, get specific campaign fields. Create, delete and update are not supported for campaigns. Campaigns data can be searched using search and enhancedsearch webservices.
- Campaigns/search- below is the sample request body.

```
{
      "searchFields" : {
         "Campaign Name" : "Test Campaign with CustomFields"
      },
      "returnFields": [
         "Campaign Name",
         "Status",
      ]
}
```

- Campaigns/enhancedsearch – Supports enhanced searching, including filters on the fields. Sample JSON for enhanced searching:

```
{
      "filters": [
{
      "comparator": "EQUAL",
      "fieldName": "Created By",
      "type": "filter",
      "values": [
         "2000000044552"
      ]
}
],
"returnFields": [
   "Campaign Name"
   ]
}
```

- New webservice has been introduced to get statistics for specific campaign.
  - getStatistics

Communication Plans

- The standard Web Services can be used to return the module details for communication plans, get specific communication plan fields. Create, delete and update are not supported for communication plans. Communication plans data can be searched using search and enhancedsearch webservices.
- CommunicationPlans/search- below is the sample JSON.

```
"searchFields": {
     "Publication Type": "General"
  },
  "returnFields": [
     "Plan Name",
        "Status"
      ]
    }
```

- CommunicationPlans/enhancedsearch – Supports enhanced searching, including filters on the fields. Sample JSON for enhanced searching:

```
    {
          "filters": [
    {
          "comparator": "EQUAL",
          "fieldName": "Created By",
          "type": "filter",
          "values": [
             "83000000145310"
          ]
    }
    ],
    "returnFields": [
       "Plan Name"
       ]
    }
```

- CommunicationPlansContacts/search- Communication plan contacts information can be fetched using this option. Below is the sample JSON.

```
    {
      "searchFields": {
         "Track ID": 2000000047003
      },

      "returnFields": [
            "Exit On",
            "Track ID",
            "Entry On",
            "Entity ID",
            "Contact ID",
            "Exit Rule",
            "Entry Rule"
      ]
    }
```

- New webservice has been introduced to get statistics for specific communication plan.
  - getStatistics

Contact Engagements

- The standard Web Services can be used to return the module details for contact engagements, get specific contact engagement data using unique id. Create, delete and update are not supported for contact engagements. Contact engagement logs can be searched using search and enhancedsearch webservices.
- ContactEngagements/search- Contact engagement logs can be searched only using contacts unique id. below is the sample JSON.

```
{
  "searchFields": {
     "Contact Name": "83000000128254"
  },
  "returnFields": [
     "Entity ID",
     "Contact Name"
  ]
}
```

- ContactEngagements/enhancedsearch – Supports enhanced searching, including filters on the fields. Sample JSON for enhanced searching:

```
{
      "filters": [
{
      "comparator": "EQUAL",
      "fieldName": "Most Recent Bounce Time",
      "type": "filter",
      "values": [
         "23/03/2021 10:04 AM"
      ]
}
],
"returnFields": [
   "Contact Name"
   ]
}
```

Contact Engagement Logs

- The standard Web Services can be used to return the module details for contact engagement logs, get specific contact engagement log data using unique id.

Create, delete and update are not supported for contact engagement logs. Contact engagement logs can be searched using search and enhancedsearch webservices.

- ContactEngagementLogs/search- below is the sample JSON.

```
{
  "searchFields": {
     "Log Type": "Campaign Sent"
  },
  "returnFields": [
     "Entity ID",
     "Contact Name"
  ]
}
```

- ContactEngagementLogs/enhancedsearch – Supports enhanced searching, including filters on the fields. Sample JSON for enhanced searching:

```
{
      "filters": [
{
      "comparator": "EQUAL",
      "fieldName": "Log Type",
      "type": "filter",
      "values": [
         "Campaign Sent"
      ]
}
],
"returnFields": [
   "Contact Name"
   ]
}
```

Usage Limits & Best Practices

Web services are a powerful tool that, when used properly, can enhance the client's efficiency and overall productivity. With its wide variety of uses and customization, the client has the ability to determine how and when the feature is utilized. As a Radius user, it is important to note the usage limits and best practices for Radius web services.

- We reserve the right to add entity properties at any time in order to enhance the product functionality. It is recommended that clients account for this during the implementation of their solution. Failure to account for this can result in the breaking of the integration after Web Services functionality is released in the future.

- Renaming Custom Fields in Radius may result in the breaking of the web service. We highly recommend using “Field ID” instead of “Field Label” in your Web Service calls. See the various Web Service methods listed above.

- The web service functionality will be rolled out in phases. Clients will be notified via release communications as new functionality is added.

- All pieces of data are case sensitive. Failure to adhere to this will result in web service errors.

- Field dependencies do not validate via web services. It is the responsibility of the client to insert proper data into those fields that have dependencies.

Product Releases & Web Services

Periodically, we schedule product releases that integrate enhancements and new features to Radius. We schedule product releases during hours that will cause the minimal amount of disruption to clients. During these times, it is important to note there will be downtime and Radius Web Services may be unavailable. Clients will need to account for downtime and accommodate for it in their code that references Radius Web Services. We do not store or queue web service calls made by clients, vendors, etc. In the event of an outage, clients need to capture the error condition, accommodate for it, and send the request at a later time when the Radius Web Services become available.

The downtime response sent back by the Radius Web Services will contain the phrase “Scheduled Maintenance Notification”. We recommend that you perform a check for this phrase in the response handling of your Web Service calls to Radius, to accommodate for the downtime scenario.

Resources

There are various resources available on the Internet. Below are some resources that provide additional information on the fundamentals of web services.

Rest Web Services Demystified
(http://www.infoworld.com/article/2614859/application-development/how-to--rest-web-servicesdemystified.html)

Building a RESTful Web Service
(http://spring.io/guides/gs/rest-service/)

JSON.org
(http://www.json.org/)

Digest Authentication
(http://en.wikipedia.org/wiki/Digest_access_authentication)

> Digest Authentication is the authentication mechanism used to authenticate against the Radius Web Services. The Web Services Username & Password are generated directly from within the Radius User Interface via the Setup screen.

Working with web services takes practice and a basic understanding of the necessary components. There are a number of web-based tools that will allow for practice. These tools can be found by conducting an Internet search. Two options (listed below) have a user interface that is simple to use.

PostMan (Application) (https://www.getpostman.com/)

Advanced REST Client (Chrome)
(https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo)

RESTClient (Firefox)
(https://addons.mozilla.org/en-US/firefox/addon/restclient/)

HurLit – Make HTTP Requests
(https://www.hurl.it/)

It is important to understand the user's Locale Setting and how this impacts Dates & Times in Web Service calls. Dates and Times are returned in the "Natural Date/Time Format" that is human-readable, based on the Locale preferences of the user specified for the Web Service. In some locales, dates/times may or may not be in the 24-hour format. If so, you won't need to pass "AM/PM" designator for the time component. If not using the 24-hour format, you may need to pass the "AM/PM" designator for the time component.

Date Format by Country
(http://en.wikipedia.org/wiki/Date_format_by_country)

Time Zones
(http://en.wikipedia.org/wiki/Time_zone)

Code Samples

The code samples below are provided as a reference only. There are numerous programming languages and libraries that can be used to interact with the Radius Web Services. The samples below use Digest Authentication using Cold Fusion, Python, & PHP.

Sample Contact Web Services using PHP:

http://apps.askadmissions.net/radius/origin/exampleCode.php (Page may need to be refreshed as you are testing the Web Services.)

Digest Authentication using Cold Fusion:

```
<!--
	See: http://en.wikipedia.org/wiki/Digest_access_authentication

	We'll need to do the following to make a Web Service request:

1.	The client asks for a page that requires authentication but does not provide a username and password.[note 2] Typically this is because the user simply entered the address or followed a link to the page.
2.	The server responds with the 401 "Unauthorized" response code, providing the authentication realm and a randomly generated, single-use value called a nonce.
3.	At this point, the browser will present the authentication realm (typically a description of the computer or system being accessed) to the user and prompt for a username and password. The user may decide to cancel at this point.
4.	Once a username and password have been supplied, the client re-sends the same request but adds an authentication header that includes the response code.		5. In this example, the server accepts the authentication and the page is returned. If the username is invalid and/or the password is incorrect, the server might return the "401" response code and the client would prompt the user again.

	Variables:

	username=supplied from Radius Web Services Create
	password=supplied from Radius Web Services Create
	method=(POST/GET/PUT/DELETE)
	digestURI=path part of URL
	realm=supplied from the authentication request
	nonce=supplied from the authentication request
```

```
        caculated values-

        A1=username:realm:password
        A2=method:digestURI
        HA1=MD5(A1)
        HA2=MD5(A2)
        response=MD5(HA1:nonce:HA2)

        IMPORTANT:        RFC 2617 (HTTP Authentication: Basic and Digest Access
Authentication) replaces the original RFC 2069, which contains optional improvements.

   The following additional values will need to be gathered and calculated if RFC 2617 is
used:

                        cnonce
                        nonceCount
                        qop

                        The Advanced Rest Client appears to use RFC 2617.

-->
<cfset username = "***YourUsername***">
<cfset password = "***YourPassword***">

<cfset method = "POST">
<cfset servername = "***<Host>***">
<cfset digestURI = "/crm/webservice/modules/Contacts/search">
<cfset URI = "https://#servername##digestURI#">

<!-- Get an authorization. -->
<cfhttp url="#URI#" method="#method#">
        <cfhttpparam type="header" name="Content-Type" value="application/json">
</cfhttp>

<cfset responseHeaders = cfhttp.ResponseHeader>
<cfset wwwAuthenticate = responseHeaders["WWW-Authenticate"] />

<!-- Parse the response and grab the challenge. -->
<cfset tempList = Replace(wwwAuthenticate,"Digest ","","one")>
<cfset authArray = ListToArray(Trim(tempList))>

<cfset challenge = StructNew()>
<cfloop from="1" to="#arrayLen(authArray)#" index="i">
```

```
        <cfset headerArr = authArray[i]>

        <cfset key = Trim(Replace(Left(headerArr,Find('=',headerArr)),'=','','ALL'))>
        <cfset value =
Trim(Replace(RemoveChars(headerArr,1,Find('=',headerArr,"1")),'"','','ALL'))>
        <cfset challenge[key] = value>
</cfloop>

<cfdump var="#challenge#">

<!-- Capture the realm and nonce. -->
<cfset realm = challenge["realm"]>
<cfset nonce = challenge["nonce"]>

<!-- Build up our digest authorization response. -->
<cfset A1 = "#username#:#realm#:#password#">
<cfset A2 = "#method#:#digestURI#">
<cfset HA1 = lcase(Hash("#A1#","MD5"))>
<cfset HA2 = lcase(Hash("#A2#","MD5"))>
<cfset response = lcase(Hash("#HA1#:#nonce#:#HA2#", "MD5"))>

<cfset authorization = 'Digest username="#username#", realm="#realm#",
nonce="#nonce#", uri="#digestURI#", response="#response#"'>

<cfset jsonRequest = "{ ""searchFields"": { ""First Name"": ""TestFN"", ""Last Name"":
""TestLN"" }, ""returnFields"": [ ""Entity ID"", ""First Name"", ""Last Name"" ] }">
<cfdump var="#jsonRequest#">

<!-- Make our final request. -->
<cfhttp url="#URI#" method="#method#">
        <cfhttpparam type="header" name="Authorization" value="#authorization#">
        <cfhttpparam type="header" name="Content-Type" value="application/json">
        <cfhttpparam type="body" value="#jsonRequest#">
</cfhttp>
<cfdump var="#cfhttp#">
```

Digest Authentication using Python “urllib2” Library:

```
class RadiusService(object):
    """
    Main wrapper for API calls.  Stores user:password and handles actual requests
    """     def
__init__(self,user,password,host):
        self.user = user
self.password = password
        self.host = host
        self.url = 'https://%s/crm/webservice/' % self.host
self.last_url = ''
        self._saved_module_list = []

        mgr = urllib2.HTTPPasswordMgrWithDefaultRealm()
mgr.add_password(None, "https://%s" % self.host, self.user, self.password)
self.opener = urllib2.build_opener(urllib2.HTTPDigestAuthHandler(mgr))

        self.saved_modules = {}
```

Digest Authentication using PHP to Search for Entities

```
<?php
$userName = "***YourUsername***";
$passKey = "***YourPassword***";
$host = "***<Host>***";
$module = $_GET['module'];
$searchFields = urldecode($_GET['searchFields']);
$returnFields = urldecode($_GET['returnFields']);

$page = $_GET['page'];
$pageSize = $_GET['pageSize'];
$queryId = $_GET['queryId'];


$search = explode("|",$searchFields);
$dataSearch = array();
for($i=0;$i<sizeof($search);$i++)
{
        $field = explode(":",$search[$i]);
        $dataSearch[$field[0]]=$field[1];
}
$dataSearch = json_encode($dataSearch);


$return = explode("|",$returnFields);
$dataReturn = json_encode($return);
```

```
$data = '{"searchFields": '.$dataSearch.',"returnFields":'.$dataReturn.'}';
//$data = $dataSearch.$dataReturn;

if($module)
{
        $call    = $host."modules/".$module."/search/";
}
else
{
        $call    = $host."modules/"."noModule"."/search/";
}

$curl = curl_init($call); curl_setopt($curl,
CURLOPT_HTTPHEADER, array('Content-Type:
application/json','Connection: Keep-Alive'));
curl_setopt($curl, CURLOPT_HTTPAUTH, CURLAUTH_DIGEST);
curl_setopt($curl, CURLOPT_USERPWD, $userName.":".$passKey);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
curl_setopt($curl, CURLOPT_POST, true); curl_setopt($curl,
CURLOPT_POSTFIELDS, $data);
$curl_response = curl_exec($curl);
echo $curl_response;
?>
```

Sample JSON for Creating a Decision Record:

```
{
   "createFields": {
      "Registration":165000001654301,
      "Decision Status":"Accept",
      "Enrollment Status":"Accepted",
      "Enrollment Form":165000001654170,
      "Decision Letter":165000001654033,
      "Publish Status":"Not Published",
      "Decision Owner":165000000620009
   },
   "returnFields": [
      "Entity ID"
   ]
}
```