Radius Web Services - Table of Contents

Page numbers are clickable.

Version Numbers

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

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, 21014

RADIUS WEB SERVICES

Summary

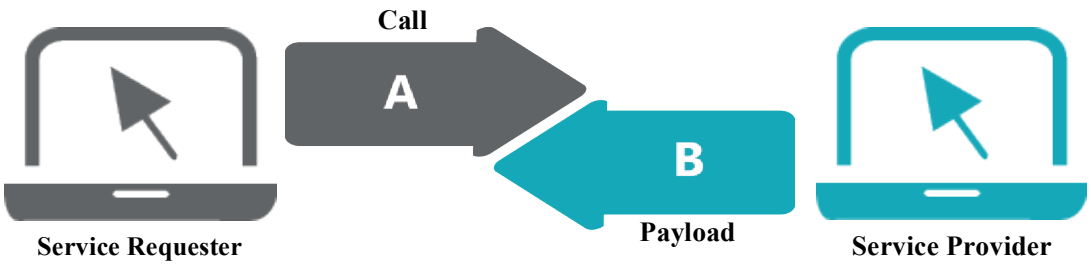
Hobsons is excited to offer native Web Services capabilities for Radius. This feature gives Radius clients the ability to customize and launch a myriad of data transfers with Radius by utilizing the Web Service API methods documented within.

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

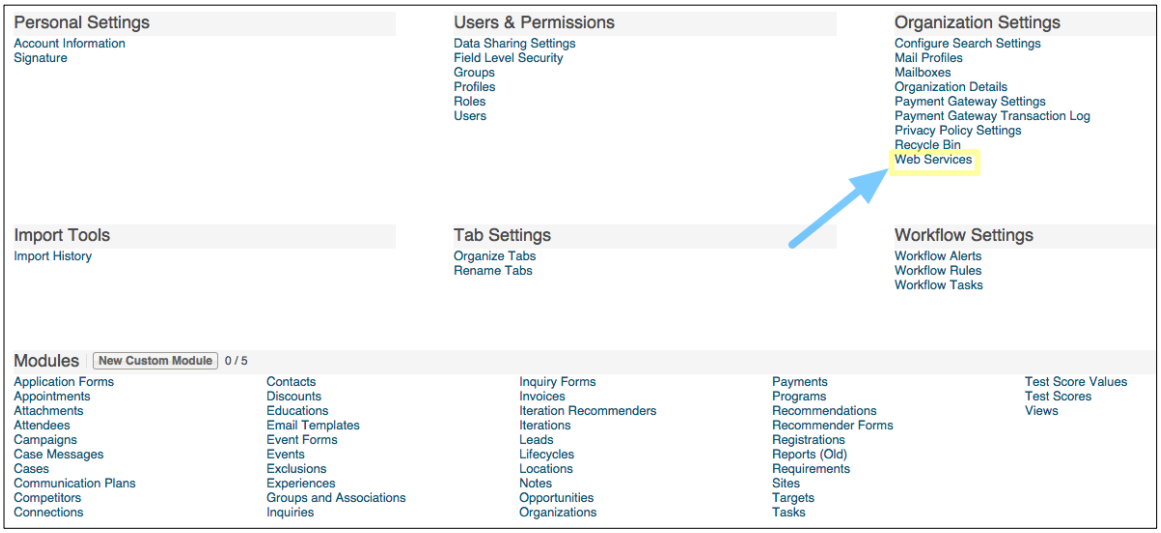
The Radius Web Service is built upon the Representational State Transfer (REST) architectural style, and utilizes JavaScript Object Notation (JSON) format.

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.

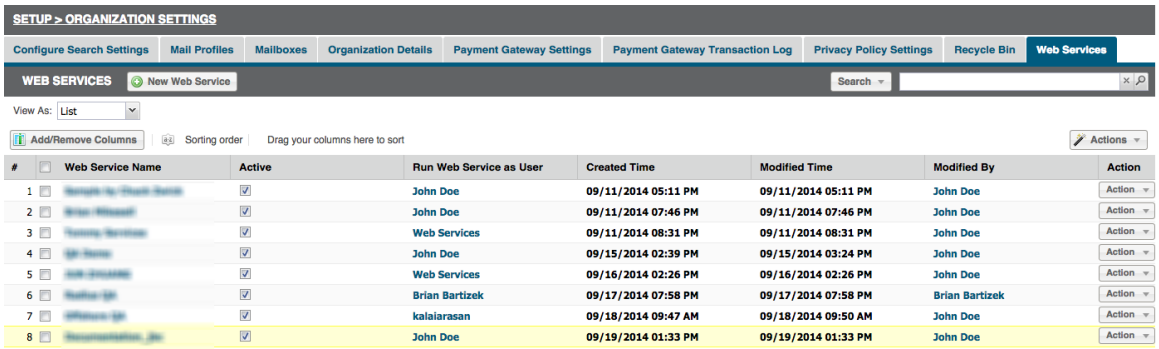
Creating a Web User Account

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.

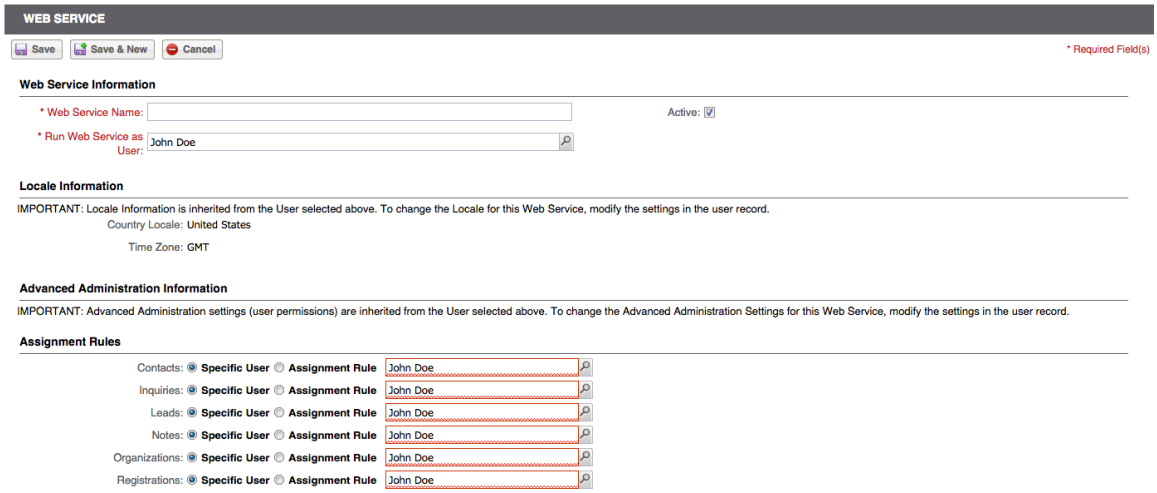
Step 1: Navigate to the Web Services
Navigate and click on **Setup**. The link can be found in the upper-right portion of the Radius user interface. **Web Services** is accessible via **Organization Settings.**



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



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.



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.

Example

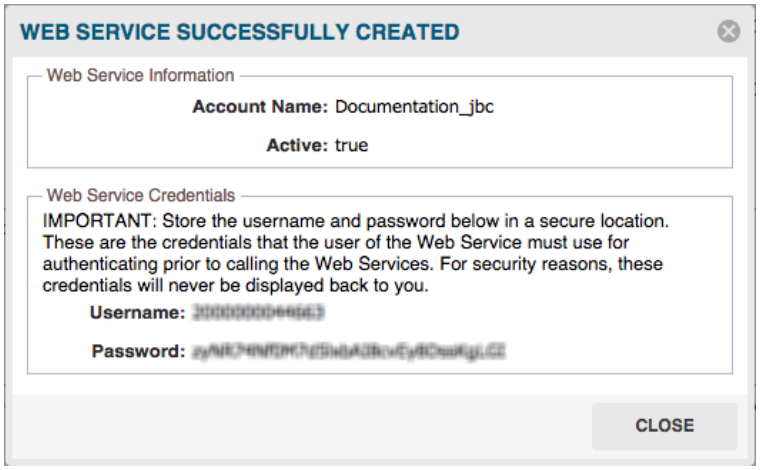
In creating a new web service account, the administrator changes the assigned user. This user has limited permissions within Radius, and therefore the web service account will inherit the permission level of the assigned user.

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.



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.**

Hobsons 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

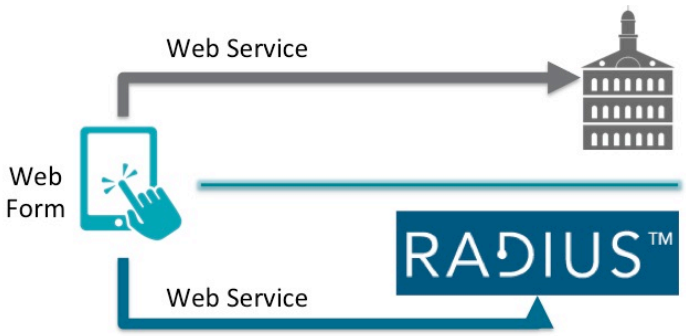
Web services are a versatile tool that can be used to integrate various systems with Radius. Every Hobsons client is unique and therefore there are numerous cases where web services can be utilized.

Here are some common examples of web services use cases:

BEST PRACTICE: If you are uncertain as to how you would use web services, or need assistance in the setup, contact your Hobsons Representative.

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",
    "Cases",
    "Programs",
    "Inquiries",
    "Venues",
    "RelatedAccounts",
    "Tasks",
    "Notes",
    "Iterations",
    "CaseMessages",
    "Appointments",
    "Registrations",
    "Leads",
    "Contacts",
    "Experiences",
    "Educations",
    "Lifecycles",
    "Recommendations",
    "Connections",
    "Users",
    "Groups",
    "Roles",
    "Profiles",
    "Decisions",
  ]
}
```

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.

<host> for US clients: api.hobsonsradius.com
<host> for EMEA clients: emeaapi.hobsonsradius.com

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

List All Modules

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

Get a Module's Meta Data

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. Hobsons highly recommends 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 when returning the entity. 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 when returning the entity. 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 created since the provided date.
- updatedSince - Returns records modified since the provided 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"
                        }
                ]
        }
}
```

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 when returning the entity
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":"","public":true,"systemDefined":"","canChangeVisibility":true,"color":""},
{"value":"b","description":"","public":true,"systemDefined":"","canChangeVisibility":true,"color":""}]},

{"groupName":"c","values":[{"value":"d","description":"","public":true,"systemDefined":"","canChangeVisibility":true,"color":""},
{"value":"e","description":"","public":true,"systemDefined":"","canChangeVisibility":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
}
}
}

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.

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.

- Hobsons reserves 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. Hobsons highly recommends 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, Hobsons will schedule product releases that integrate enhancements and new features to Radius. Hobsons schedules 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 down time and Radius Web Services may be unavailable. Clients will need to account for down time and accommodate for it in their code that references Radius Web Services. Hobsons does 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.

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-services-demystified.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.

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.

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.

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"
   ]
}
```