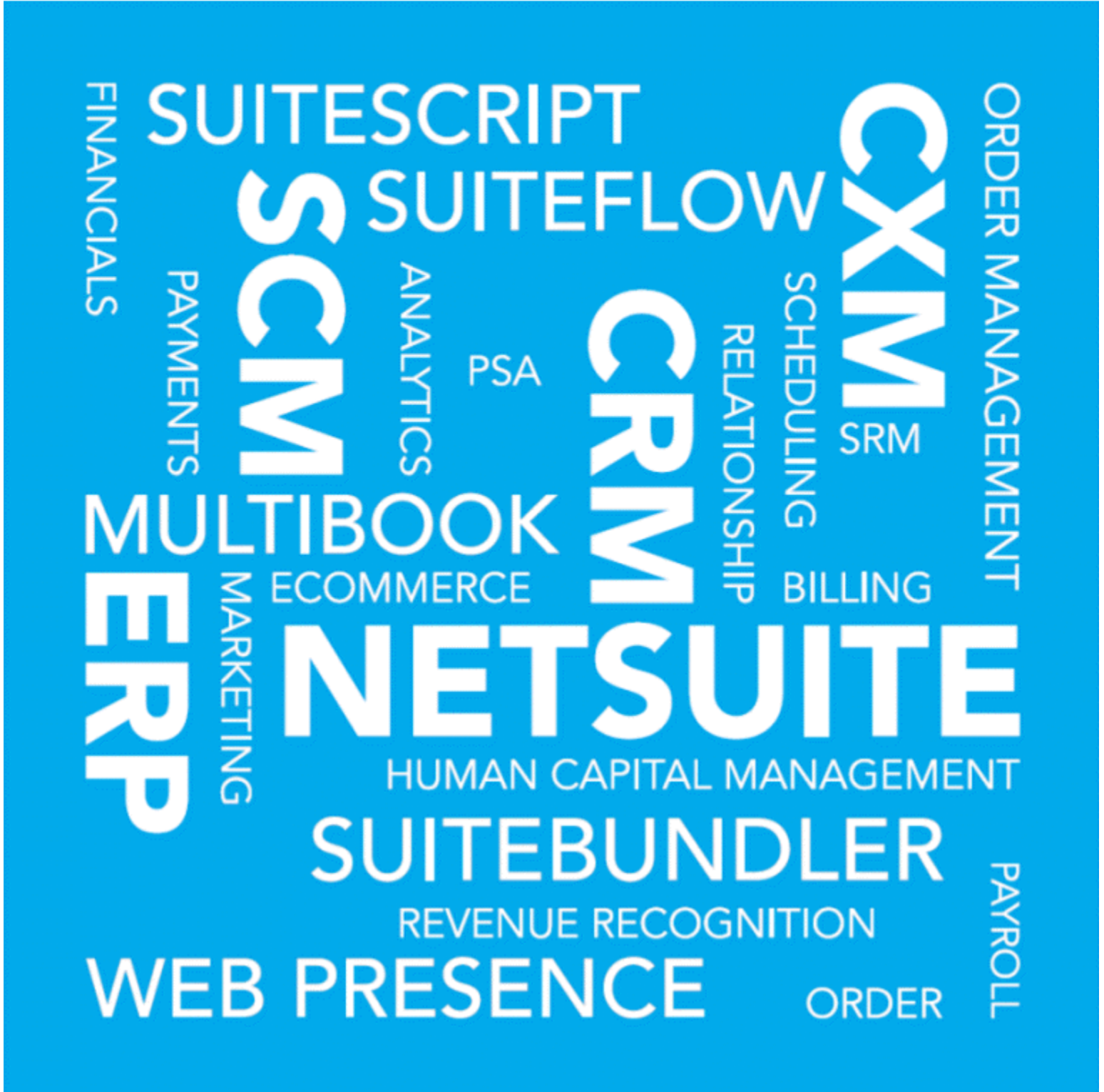


# SuiteTalk REST Web Services



Copyright © 2005, 2019, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

If this document is in public or private pre-General Availability status:

This documentation is in pre-General Availability status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

If this document is in private pre-General Availability status:

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your pre-General Availability trial agreement only. It is not a commitment to deliver any material, code, or functionality, and

should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Master Agreement, Oracle License and Services Agreement, Oracle PartnerNetwork Agreement, Oracle distribution agreement, or other license agreement which has been executed by you and Oracle and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

### **Sample Code**

Oracle may provide sample code in SuiteAnswers, the Help Center, User Guides, or elsewhere through help links. All such sample code is provided "as is" and "as available", for use only with an authorized NetSuite Service account, and is made available as a SuiteCloud Technology subject to the SuiteCloud Terms of Service at [www.netsuite.com/tos](http://www.netsuite.com/tos).

Oracle may modify or remove sample code at any time without notice.

### **No Excessive Use of the Service**

As the Service is a multi-tenant service offering on shared databases, Customer may not use the Service in excess of limits or thresholds that Oracle considers commercially reasonable for the Service. If Oracle reasonably concludes that a Customer's use is excessive and/or will cause immediate or ongoing performance issues for one or more of Oracle's other customers, Oracle may slow down or throttle Customer's excess use until such time that Customer's use stays within reasonable limits. If Customer's particular usage pattern requires a higher limit or threshold, then the Customer should procure a subscription to the Service that accommodates a higher limit and/or threshold that more effectively aligns with the Customer's actual usage pattern.

### **Beta Features**

Oracle may make available to Customer certain features that are labeled "beta" that are not yet generally available. To use such features, Customer acknowledges and agrees that such beta features are subject to the terms and conditions accepted by Customer upon activation of the feature, or in the absence of such terms, subject to the limitations for the feature described in the User Guide and as follows: The beta feature is a prototype or beta version only and is not error or bug free and Customer agrees that it will use the beta feature carefully and will not use it in any way which might result in any loss, corruption or unauthorized access of or to its or any third party's property or information. Customer must promptly report to Oracle any defects, errors or other problems in beta features to [support@netsuite.com](mailto:support@netsuite.com) or other designated contact for the specific beta feature. Oracle cannot guarantee the continued availability of such beta features and may substantially modify or cease providing such beta features without entitling Customer to any refund, credit, or other compensation. Oracle makes no representations or warranties regarding functionality or use of beta features and Oracle shall have no liability for any lost data, incomplete data, re-run time, inaccurate input, work delay, lost profits or adverse effect on the performance of the Service resulting from the use of beta features. Oracle's standard service levels, warranties and related commitments regarding the Service shall not apply to beta features and they may not be fully supported by Oracle's customer support. These limitations and exclusions shall apply until the date that Oracle at its sole option makes a beta feature generally available to its customers and partners as part of the Service without a "beta" label.

## Send Us Your Feedback

We'd like to hear your feedback on this document.

Answering the following questions will help us improve our help content:

- Did you find the information you needed? If not, what was missing?
- Did you find any errors?
- Is the information clear?
- Are the examples correct?
- Do you need more examples?
- What did you like most about this document?

Click [here](#) to send us your comments. If possible, please provide a page number or section title to identify the content you're describing.

To report software issues, contact NetSuite Customer Support.

# Table of Contents

SuiteTalk REST Web Services Overview and Setup .....	1
REST Web Services Key Concepts .....	1
REST Web Services and Other Integration Options .....	2
REST Web Services Prerequisites and Setup .....	3
REST Web Services URL Schema and Account-Specific URLs .....	5
Authentication for REST Web Services .....	8
Working with REST Web Services Using Postman .....	10
Importing and Setting Up a Postman Environment .....	11
Importing the Postman Collection .....	12
Sending a Request From the Imported Collection .....	12
Working with Resource Metadata .....	14
Getting Metadata .....	15
Working with OpenAPI 3.0 Metadata .....	16
Working with JSON Schema Metadata .....	20
Working with Records .....	23
The REST API Browser .....	23
NetSuite Record Structure .....	25
Using CRUD Operations on Custom Records .....	26
Creating a Record Instance .....	26
Getting a Record Instance .....	27
Format of Sublists and Subrecords .....	28
Format of Selects and References .....	29
Format of Multiselects .....	30
Format of Enumeration Values .....	30
Updating a Record Instance .....	30
Using the Upsert Operation .....	31
Deleting a Record Instance .....	31
Using External IDs .....	31
Using Datetime Fields .....	32
Executing Record Actions .....	33
Transforming Records .....	34
Using the REST Web Services SuiteScript Execution Context .....	35
Working with Sublists .....	37
Creating a Sublist .....	38
Updating a Sublist .....	39
Replacing a Sublist .....	41
Working with Subrecords .....	47
Record Filtering and Query .....	49
Listing All Record Instances .....	49
Record Collection Filtering .....	50
Executing SuiteQL Queries Through REST Web Services .....	52
Working with SuiteAnalytics Workbooks in REST Web Services .....	53
Collection Paging .....	55
Error Handling and Logging in REST Web Services .....	58
Error Handling in REST Web Services .....	58
Using the REST Web Services Execution Log .....	59

# SuiteTalk REST Web Services Overview and Setup

The NetSuite REST web services provide an integration channel that extends the capabilities of SuiteTalk. REST web services provide a REST-based interface for interacting with NetSuite.

Using REST web services beta version, you can:

- Use CRUD (create, read, update, delete) operations to perform business processing on NetSuite records and to navigate dynamically between records. For details, see [Working with Records](#).
- Get and process the API definition and record metadata. For details, see [Working with Resource Metadata](#).
- Execute NetSuite queries on records. For details, see [Record Filtering and Query](#).

## Benefits of REST Web Services

The main benefits of REST web services include the following:

- Simple access to records metadata. This includes user and company-specific metadata. For more information about working with records metadata, see [Working with Resource Metadata](#).
- Easier handling of custom records and custom fields.
- Easy to navigate API.
- In contrast to RESTlets, you do not need to write, deploy, and run custom scripts.

## Limitations REST Web Services

Consider the following limitations when working with REST web services.

- Enumeration values must be provided and are returned in the SuiteScript internal format. This is the format used by `Record.getValue(options)` and `Record.setValue(options)` functions. For information about using these methods, see the [SuiteScript Developer Guide](#).
- Query only returns record IDs and HATEOAS links. (See [HATEOAS](#).) That is, query results have a form of non-expanded references. Additionally, you can only use body fields in query conditions.
- Making queries on transactions and certain other record types is not supported.

For more information, see the following topics:

- [REST Web Services Key Concepts](#)
- [REST Web Services and Other Integration Options](#)
- [REST Web Services Prerequisites and Setup](#)
- [REST Web Services URL Schema and Account-Specific URLs](#)

## REST Web Services Key Concepts

The following sections introduce the main concepts of REST web services.

## HATEOAS

Hypermedia as the Engine of Application State is an essential principle that should be respected in RESTful APIs.

In practice this means that you can navigate to the referenced resources without deeper knowledge of the system. A typical response contains "links" sections for each resource, which can be a sub-resource of a parent resource or any other referenced resource. You can use links to work with those resources.

For example, when getting sales order data, the response contains a customer reference field that contains a links section. You can then use the link to get data of the particular customer.

For more information, see the following resources:

- <https://en.wikipedia.org/wiki/HATEOAS>
- [https://en.wikipedia.org/wiki/Link\\_relation](https://en.wikipedia.org/wiki/Link_relation)
- <https://tools.ietf.org/html/rfc5988>

## Resource

A resource represents some data which can be uniquely identified. Each resource has its own unique URL, and each resource can reference other resources.

The two main types of resources are the following:

- Singular resources
- Collection resources that contain multiple singular resources

Resources can exist in hierarchy, and can form a tree structure, consisting of child and parent resources.

In NetSuite, the most important resource is a record. A record is a singular resource. However, there can be other resources in NetSuite as well.

A record usually references other resources - other records.

An example of a collection resource is a sublist because it contains multiple lines. Each line is a singular child resource, and the record is a parent resource.

A record with multiple sublists, each of them with multiple lines, forms a hierarchical resource.

## REST Web Services and Other Integration Options


To decide the best integration option for your purposes, consider the following comparisons.

The following table compares the characteristics of REST web services with those of SOAP web services and RESTlets.

**Note:** The limits for concurrent requests are unified for SOAP and REST web services and for RESTlets. For information about account concurrency, see the help topic [Web Services and RESTlet Concurrency Governance](#).

**Note:** For a comparison of RESTlets with other integration options, see the help topic [RESTlets vs. Other NetSuite Integration Options](#).

Attribute	REST Web Services	SOAP Web Services	RESTlets
Supported Operations	get, search, add, update, delete	get, search, add, update, delete	get, search, add, update

Attribute	REST Web Services	SOAP Web Services	RESTlets
Authentication Supported?	Yes (token-based authentication)	Yes (user credentials; token-based authentication)	Yes (user credentials; token-based authentication)
Passing of Login Details	in OAuth authorization header	in body (SOAP)	in authorization header
Passing of Parameters	all parameters on URL	all parameters in body (SOAP)	GET parameters on URL
Supported Content Types	JSON, Swagger	text/xml (explicit)	JSON, text/xml (explicit)
Environment	lightweight, no coding and script deployment needed on the server side, suitable for mobile devices	heavy programming and deployment environment (C#, Java)  no coding and script deployment needed on the server side	lightweight, suitable for mobile devices, bundleable
Functionality	<ul style="list-style-type: none"> <li>■ Metadata catalog. See <a href="#">Working with Resource Metadata</a></li> <li>■ CRUD. See <a href="#">Working with Records</a>.</li> <li>■ <a href="#">Record Filtering and Query</a></li> </ul>	<ul style="list-style-type: none"> <li>■ CRUD</li> <li>■ Search</li> </ul>	<p>All functionality available through SuiteScript:</p> <ul style="list-style-type: none"> <li>■ CRUD</li> <li>■ Search</li> <li>■ Query</li> </ul> <div style="border: 1px solid #0070c0; padding: 5px; margin-top: 10px;"> <p> <b>Note:</b> SuiteScript does not provide metadata.</p> </div>
Standards	Similar to the REST APIs of other Oracle products	SOAP protocol	No standards
Required User Knowledge	REST programmer API level knowledge	SOAP programmer API level knowledge	JavaScript programmer SuiteScript level knowledge
Performance	Using REST API, fewer calls may be required to accomplish a business flow. Therefore the overall performance may be better than SOAP and CSV.	SOAP web services require more calls to accomplish a business flow than the REST API.	RESTlets are the fastest integration channel. All actions required for a business flow can be executed within a single call.


## REST Web Services Prerequisites and Setup

To use REST web services, the relevant features must be enabled in your account. Additionally, the REST web services user must have the required permissions assigned to the user's role. See the following table for information about enabling the required features and assigning the required permissions.

Feature or Permission	Usage
REST Web Services feature	Enable the feature at Setup > Company > Setup Tasks > Enable Features, in the SuiteTalk (Web Services) section, on the SuiteCloud subtab. To use the feature, you must accept the SuiteCloud Terms of Service.



Feature or Permission	Usage
SuiteAnalytics Workbook feature	Enable the feature at Setup > Company > Setup Tasks > Enable Features, on the Analytics subtab.
Permissions: <ul style="list-style-type: none"> <li>■ REST Web Services</li> <li>■ Log in using Access Tokens</li> <li>■ SuiteAnalytics Workbook</li> </ul>	See <a href="#">Standard Roles with the REST Web Services, SuiteAnalytics Workbook, and Log in Using Access Tokens Permissions</a> for a list of roles that have the required permissions assigned by default.  See <a href="#">Assigning the Required Permissions to a User's Role</a> for information about assigning the permissions manually.

 **Note:** Using the administrator role for building web services integrations is not recommended.

## Standard Roles with the REST Web Services, SuiteAnalytics Workbook, and Log in Using Access Tokens Permissions

The following standard roles have the SuiteAnalytics Workbook, Log in using Access Tokens, and REST Web Services permissions assigned by default. For more information about standard roles, see the help topic [NetSuite Roles Overview](#).

- Accountant
- Accountant (Reviewer)
- A/P Clerk
- A/R Clerk
- Bookkeeper
- CEO (Hands Off)
- CEO
- Sales Manager
- Sales Person
- Store Manager
- Support Manager
- Support Person
- Employee Center
- Warehouse Manager
- Payroll Manager
- Partner Center
- Intranet Manager
- Marketing Manager
- Marketing Assistant
- System Administrator
- Sales Administrator
- Support Administrator
- Marketing Administrator
- Advanced Partner Center

- Online Form User
- Engineer
- Issue Administrator
- Payroll Setup
- Engineering Manager
- Product Manager
- CFO
- Retail Clerk (Web Services Only)
- Buyer
- Developer

### Assigning the Required Permissions to a User's Role

If you use a custom role, or a standard role that does not have the required permissions by default, you can assign the permissions manually. To assign the required permissions to a role:

1. Go to Setup > Users/Roles > User Management > Manage Roles.
2. Locate the role you want to modify. Click the corresponding **Edit** or **Customize** link.
3. On the Permissions subtab, click Setup.
4. In the Permission list, select REST Web Services.
5. In the Level list, select Full.
6. Click Add.
7. On the Permissions subtab, click Setup.
8. In the Permission list, select Log in using Access Tokens.
9. In the Level list, select Full.
10. Click Add.
11. On the Permissions subtab, click Reports.
12. In the Permission list, select SuiteAnalytics Workbook.
13. In the Level list, select Full.
14. Click Add.
15. Click Save.

## REST Web Services URL Schema and Account-Specific URLs

You can only access REST web services using account-specific domains. Account-specific domains are unique to your account because they contain your account ID as part of the domain name. These domains do not change when your account is moved to a different data center.

When using account-specific domains, dynamic domain discovery is not needed.

The format of an account-specific domain name is the following: <account ID>.<service>.netsuite.com. For example, if your account ID is 123456, your account-specific domain for REST web services is: 123456.suitetalk.api.netsuite.com

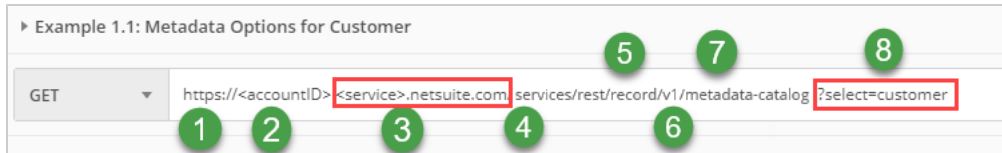
You can find the service URL for SuiteTalk (SOAP and REST web services) at Setup > Company > Setup Tasks > Company Information, on the Company URLs subtab. When you build a client application, you must ensure that the service URL is a configurable parameter.



**Important:** The examples in this document use demo123 as the account ID. Substitute this with your own account ID.

You can access NetSuite resources through REST web services by using URLs specific to either record or query service.

A URL for a REST resource looks like the following.



The URL consists of the following parts:

1. Protocol
2. The account ID of your NetSuite account
3. The domain name for REST web services
4. The complete REST API services endpoint
5. The specific services that is being accessed, for example, record, query, or other service
6. The service version, which specific to each service
7. The optional sub-service, for example, **metadata-catalog**, **suiteql**, or **workbook**
8. Query parameters

The following examples demonstrate the usage of the record and query services.

To use CRUD operations on records, you can use a URL similar to the following:

```
PATCH https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/42
```

For information about CRUD operations, see [Working with Records](#).

**Metadata-catalog** is a subservice of the record service. To access resource metadata, you can use a URL similar to the following:

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/?select=customer
```

For information about using metadata, see [Working with Resource Metadata](#).

**Workbook** is a subservice of the query service. To execute saved workbooks, you can use a URL similar to the following:

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/query/v1/workbook/custworkbook123/result
```

For information about saved datasets, see [Working with SuiteAnalytics Workbooks in REST Web Services](#).

**Suiteql** is a subservice of the query service. To execute SuiteQL queries, you can use a URL similar to the following:

```
POST https://demo123.suitetalk.api.netsuite.com/services/rest/query/v1/suiteql?limit=10&offset=10
```

For information about SuiteQL queries in REST web services, see [Executing SuiteQL Queries Through REST Web Services](#).

# Authentication for REST Web Services

NetSuite supports two mechanisms that increase overall system security. Token-based authentication (TBA) and OAuth 2.0, a robust, industry standard-based mechanism. These authentication mechanisms enable client applications to use a token to access NetSuite through APIs, eliminating the need for integrations to store user credentials.

The REST web services beta also uses TBA for user login. There is no support for login through user credentials. For general information about TBA, see the help topic [Token-based Authentication \(TBA\)](#).

For general information about OAuth 2.0, see the help topic [OAuth 2.0](#).

## Setting Up Token-Based Authentication (TBA) for REST Web Services

To set up TBA, see the help topic [Getting Started with Token-based Authentication](#).

To be able to use REST web services with TBA, you must create an application using an integration record. See the help topic [Create Integration Records for Applications to Use TBA](#). As the last step of this procedure, make sure you note the consumer key and consumer secret.

After creating the integration application, continue with creating tokens for your users. Issue a new token for at least one of your users, and note its token ID and token secret. For details, see the help topic [Access Token Management – Create and Assign a TBA Token](#).

For detailed information about using TBA in integration applications, see the help topic [The Three-Step TBA Authorization Flow](#).

## Setting Up OAuth 2.0 Authentication for REST Web Services

To set up OAuth 2.0, see the help topic [Getting Started with OAuth 2.0](#).

To be able to use REST web services with OAuth 2.0, you must create an application using an integration record. See the help topic [Create Integration Records for Applications to Use OAuth 2.0](#). As the last step of this procedure, make sure you note the client ID and client secret.

After you create the integration application, continue with a set up of an application for use with OAuth 2.0. For more information, see the help topic [OAuth 2.0 for Integration Application Developers](#). After the application is set up, you can manage the authorized applications. For more information, see the help topic [Managing OAuth 2.0 Authorized Applications](#).

## Concurrency Governance

To optimize NetSuite application and database servers, the system employs certain mechanisms to control the consumption of web services.

Concurrency for REST web services is governed in a way that each request counts towards the account limit. The account governance limit applies to the combined total of web services and RESTlet requests.

For detailed information about concurrency governance, see the help topic [Web Services and RESTlet Concurrency Governance](#).

Additionally, if a request takes more than 15 minutes to complete, it automatically times out.

These mechanisms ensure the following:

- Requests are monitored and controlled to ensure that the user experience is not excessively impacted.
- The burden of heavy web services users is not shared among all users.


# Working with REST Web Services Using Postman

In this document, the REST web services functionality is demonstrated using the [Postman Application](#). However, you can use any similar tool of your preference to work with REST web services. Besides being able to build and send the API requests, the Postman Application also acts as a library for your requests which you can then import and export. You can download Postman at <https://www.getpostman.com/>

## Installing Postman

Follow these steps to install Postman:

1. Download and install the Postman desktop application from <https://www.getpostman.com/>.

 **Note:** Do not use the (deprecated) Chrome extension, as it does not support some of the features that are present in the REST web services sample request collection.


2. Run Postman. Click Take me straight to the app on the initial splash screen.
3. Close the initial task window.

## Working with Postman Environments and Collections

A Postman environment is a set of key-value pairs. The key represents the name of the variable. Using a Postman environment, you can switch between various NetSuite accounts, and between your test or production accounts. Using Postman environments, you can customize requests using variables so you can switch between different setups without changing your requests. You can also download environments, save them as JSON files, and upload them later.

A Postman collection is a set of HTTP requests. Similarly to the environments, you can create, share, duplicate, export, and delete a collection. You can also import a collection as a single JSON file. The collection distributed together with this document requires a proper environment setup as described in [Importing and Setting Up a Postman Environment](#).

You can download the REST API Postman environment template and collection of sample requests from the SuiteTalk tools download page at <https://<accountID>.app.netsuite.com/app/external/integration/integrationDownloadPage.nl>. To access the page, you must substitute your account ID in the URL.

 **Note:** To access the Postman environment template and collection, the REST Web Services feature must be enabled, and you must have the REST web services permission assigned to your role. For more information, see [REST Web Services Prerequisites and Setup](#).

The set of sample requests for the Postman Application is provided to demonstrate how to use NetSuite's REST web services. The sample requests can also help you start building your REST-based integration with NetSuite.

For more information, see the following topics:

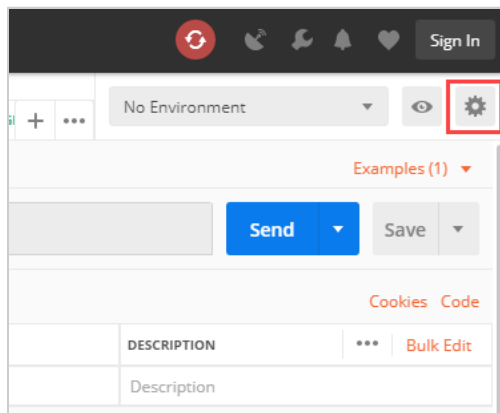
- [Importing and Setting Up a Postman Environment](#)

- Importing the Postman Collection
- Sending a Request From the Imported Collection

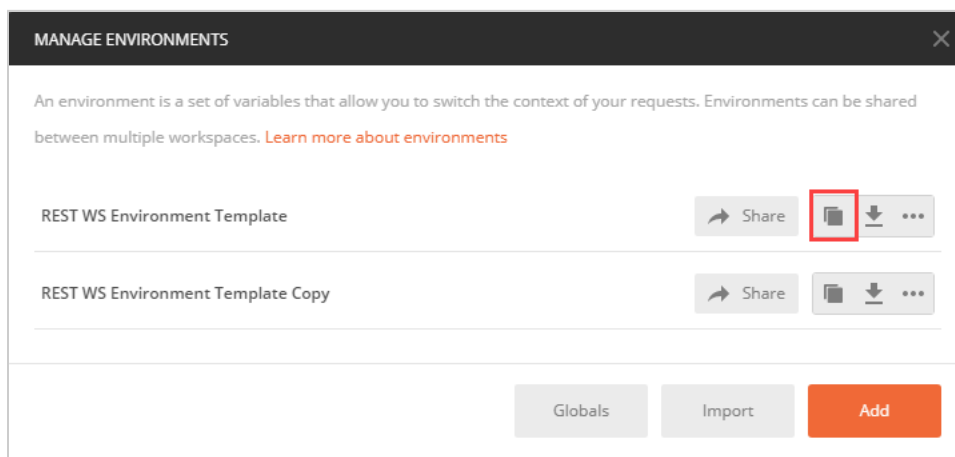
## Importing and Setting Up a Postman Environment

To import a Postman collection:

1. Download the Postman environment template and collections archive from the SuiteTalk tools download page at <https://<accountID>.app.netsuite.com/app/external/integration/integrationDownloadPage.nl>. To access the page, you must substitute your account ID in the URL. To access the page, your role must have the REST web services permission assigned to it.
2. Unzip the archive.
3. To import the environment template from the Environment folder, click the gear icon in the top menu of the Postman application.



4. A popup window opens. Click Import on this window, and then select the template file to import.
5. After importing the environment, the REST WS Environment Template is displayed in the list of environments. Make a copy of the template for each new TBA token you want to add. Click the Duplicate button to make a copy.



6. Enter a self-descriptive name for your environment, for example, "<role> for <account> at <host>". REST web services only support account-specific domains. For information about account-specific domains, see the help topic [URLs for Account-Specific Domains](#).
7. Enter your account ID (for example, 3604360).



8. Enter the TBA credentials you created. For details about creating TBA credentials, see [Authentication for REST Web Services](#).
9. Enter your company URL for SuiteTalk. You can find your company URL at Setup > Company > Setup Tasks > Company Information, on the Company URLs subtab.

MANAGE ENVIRONMENTS
✕

Environment Name

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	⋮	
<input checked="" type="checkbox"/>	ACCOUNT_ID	<ACCOUNT_ID>	<ACCOUNT_ID>		
<input checked="" type="checkbox"/>	CONSUMER_KEY	<CONSUMER_KEY>	<CONSUMER_KEY>		
<input checked="" type="checkbox"/>	CONSUMER_SECRET	<CONSUMER_SECRET>	<CONSUMER_SECRET>		
<input checked="" type="checkbox"/>	TOKEN_ID	<TOKEN_ID>	<TOKEN_ID>		
<input checked="" type="checkbox"/>	TOKEN_SECRET	<TOKEN_SECRET>	<TOKEN_SECRET>		
<input checked="" type="checkbox"/>	COMPANY_URL	<COMPANY_URL>	<COMPANY_URL>		
<input checked="" type="checkbox"/>	REST_SERVICES	{{COMPANY_URL}}/se...	{{COMPANY_URL}}/services/rest		
	Add a new variable				

Cancel
Update

## Importing the Postman Collection

To import the Postman collection:

1. Download the REST web services Postman collection of sample requests from the SuiteTalk tools download page at <https://<accountID>.app.netsuite.com/app/external/integration/integrationDownloadPage.nl>. To access this page, you must substitute your account ID, and the REST web services permission must be assigned to your role.
2. Click Import in the top menu of the Postman Application, and select the previously downloaded collection (with \*.json file extension).
3. Click the Collections tab in the left panel of the Postman Application to see the newly imported request collection.

## Sending a Request From the Imported Collection

Send a test request to verify that your Postman environment configuration and collection import is correct.

To send a test request:

1. Click the label (NetSuite REST API Tutorial) of the newly imported collection. A list of requests is displayed.
2. Open the 0 Test folder in the imported collection, and select Example 0: Test Request from the list.

3. Click Send to execute the test request. The response to the request is displayed in the Response section of the Postman window.

# Working with Resource Metadata

The REST API consists of a dynamic schema that is described by the metadata catalog. The metadata catalog serves as the API schema defining the contract, similarly to WSDL and XSD files in SOAP web services.

Using the metadata catalog, you can dynamically discover the API, including all available resources, the format and values of input and output, the supported HTTP methods and HTTP query parameters.

In NetSuite, records are the most important resource. The metadata catalog defines all available records, their fields, available values (for example, for enum fields), sublists and subrecords (both standard and custom ones), and their various properties. As new customizations are added to the system, they appear in the metadata catalog as well.

The metadata catalog for the beta version describes the following properties of the REST API:

- Names of records, fields, sublists, and subrecords
- Field types
- Reference fields and the referenced record types
- Enum fields and their available internal values
- Searchability of records, fields, sublists, and subrecords

The REST API comes with a fully personalized view on the resources (per user). This includes the ability to transparently work with user-specific NetSuite record customizations, such as custom records and fields. The ability to provide a record in its customized form means that the record structure can vary based on your specific NetSuite setup. Therefore, the REST API provides an option to dynamically generate metadata about the records (available resources and operations) in the form of standardized descriptions. Custom record metadata is accessible the same way as metadata for standard records.

Using metadata information, you can:

- Get an overview of all available record types
- Get an overview of the structure of a particular record type
- Get an overview of searchable fields
- Get an overview of supported HTTP methods and query parameters
- Automatically generate client code, for example, API client libraries or client stubs.

The beta version can provide metadata in [Swagger \(OpenAPI 3.0\)](#) and [JSON Schema](#) JSON-based formats. Both formats are used to define the structure of JSON data for validation, documentation, and interaction control.

The main difference between the two formats is that the metadata provided in JSON Schema format only describes the internal structure of a resource, for example, its fields, sublists, or subrecords.

In addition to this, the metadata in OpenAPI 3.0 format also describes links to related resources. It also describes how to interact with a resource through REST web services: it describes the URLs, HTTP methods, and parameters.

For more information, see the following help topics:

- [Getting Metadata](#)
- [Working with OpenAPI 3.0 Metadata](#)

- Working with JSON Schema Metadata

## Getting Metadata

The endpoint for getting the metadata schema for all exposed records is `http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog` where `services/rest` is the name of the REST API endpoint, `record` is the name of the service we are trying to access, `v1` is the service version, and `metadata-catalog` is the sub-resource, that is, the record metadata.

**Note:** For detailed information about the URL schema for REST web services, see [REST Web Services URL Schema and Account-Specific URLs](#).

The following is an example of a request that returns metadata about all records.

```
GET http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog
```

The following is an example of an intentionally shortened response for the previous request:

```
{
  "items": [
    {
      "name": "account",
      "links": [
        {
          "rel": "canonical",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/account",
          "mediaType": "application/json"
        },
        {
          "rel": "alternate",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/account",
          "mediaType": "application/swagger+json"
        },
        {
          "rel": "alternate",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/account",
          "mediaType": "application/schema+json"
        }
      ],
      "rel": "describes",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/account"
    }
  ]
}
```

The response informs you through HATEOAS links about the possible `mediaType` flavor in which the response can be obtained. You can see that the metadata for each record can be served in both OpenAPI 3.0 and JSON schema formats.

## Working with OpenAPI 3.0 Metadata

To get the metadata in OpenAPI 3.0 format, you have to specify the proper value in the Accept HTTP header. For OpenAPI 3.0, you must add a header in the format **Accept: application/swagger+json**.

To avoid the long loading time needed to gather OpenAPI 3.0 metadata for all records, the metadata catalog resource supports the select query parameter for selecting particular record types. You can use this parameter to restrict the metadata to certain record types only. In the following example, the request is modified to restrict the metadata to customer and sales order records only.

```
SET-HEADER Accept: application/swagger+json
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog?select=customer,salesorder
```

The following is an intentionally shortened response to the previous request.

```
"paths": {
  "/salesorder": {
    "get": {
      "tags": [
        "salesorder"
      ],
      "summary": "Get list of records",
      "parameters": [
        {
          "name": "q",
          "in": "query",
          "description": "Search query used to filter results",
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Get record",
          "content": {
            "application/vnd.oracle.resource+json; type=collection": {
              "schema": {
                "$ref": "#/components/schemas/salesorderCollection"
              }
            }
          }
        },
        "default": {
          "description": "Error response.",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/rest_error"
              }
            }
          }
        }
      }
    }
  }
}
```

```

    },
  },
}

```

The response describes the structure of the records as well as the supported operations, HTTP methods, and query parameters.

The following image shows an excerpt from the response. It describes the record name, the query parameters, and the available HTTP methods you can use with the record.

```

"paths": {
  "/customer": {
    "get": {
      "tags": [
        "salesorder"
      ],
      "summary": "Get list of records",
      "parameters": [
        {
          "name": "q",
          "in": "query",
          "description": "Search query used to filter results",
          "required": false,
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
                  "type": "object"
                }
              }
            }
          }
        }
      }
    },
    "post": {
      "tags": [
        "salesorder"
      ],
      "summary": "Create a new record",
      "parameters": [
        {
          "name": "q",
          "in": "query",
          "description": "Search query used to filter results",
          "required": false,
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object"
              }
            }
          }
        }
      }
    }
  },
  "/customer/{id}": {
    "get": {
      "tags": [
        "salesorder"
      ],
      "summary": "Get record by ID",
      "parameters": [
        {
          "name": "id",
          "in": "path",
          "description": "ID of the record",
          "required": true,
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object"
              }
            }
          }
        }
      }
    }
  },
  "/salesorder/{id}": {
    "get": {
      "tags": [
        "salesorder"
      ],
      "summary": "Get record by ID",
      "parameters": [
        {
          "name": "id",
          "in": "path",
          "description": "ID of the record",
          "required": true,
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "content": {
            "application/json": {
              "schema": {
                "type": "object"
              }
            }
          }
        }
      }
    }
  }
}

```

The following excerpt describes the record structure and the properties of different fields: their name, type, format, and attributes.

```

"salesOrder": {
  "type": "object",
  "properties": {
    "defaultILShipMethKey": {
      "type": "integer",
      "format": "int64",
      "nullable": true
    },
    "startDate": {
      "title": "Start Date",
      "type": "string",
      "description": "Sets the date for creating the first invoice. This should only be set if Advanced Billing is being used.",
      "format": "date",
      "nullable": true
    },
    "discountIsTaxable": {
      "type": "boolean"
    },
    "oppcreatedfromtitle": {
      "type": "string",
      "nullable": true
    },
    "clearedDate": {
      "title": "Date Cleared",
      "type": "string",
      "format": "date",
      "nullable": true
    }
  }
}

```

The following excerpt describes a sublist of the sales order record.

```

"salesOrder-giftCertRedemptionElement": {
  "type": "object"
}
"properties": {
  "links": {
    "title": "Links",
    "type": "array",
    "readOnly": true,
    "items": {
      "$ref": "#/components/schemas/nsLink"
    }
  },
  "authCodeApplied": {
    "title": "Amount Applied",
    "type": "number",
    "format": "double",
    "nullable": true
  },
  "authCodeAmtRemaining": {
    "title": "Available Credit",
    "type": "number",
    "format": "double",
    "nullable": true
  },
  "giftCertAvailable": {
    "type": "number",
    "format": "double",
    "nullable": true
  },
  "parentTransaction": {
    "$ref": "#/components/schemas/salesOrder"
  },
  "authCode": {
    "$ref": "#/components/schemas/nsResource"
  }
}
},

```

The OpenAPI 3.0 record metadata received in the previous step can be used as input for any OpenAPI 3.0 compatible tool for further processing. For example, you can use the metadata in tools that generate REST client stubs as well as tools that specialize in dumping a static webpage that describes record structure. In this example, OpenAPI 3.0 metadata is used in the freely available online [Swagger Editor](#) for exploring the structure and operations that can be done with the customer and sales order records.

**Note:** The performance of the online Swagger editor can be limited if there is a large amount of data to process. Therefore it is not recommended to use the editor to generate metadata description for all exposed records.

By copy-pasting the example response to the Swagger Editor, you can see an output similar to the following:

SalesOrder (Beta)		
GET	/salesOrder	Get list of records.
POST	/salesOrder	Insert record.
GET	/salesOrder/{id}	Get record.
PUT	/salesOrder/{id}	Insert or update record.
DELETE	/salesOrder/{id}	Remove record.
PATCH	/salesOrder/{id}	Update record.
POST	/salesOrder/{id}/!transform/fulfillmentRequest	Transform to fulfillmentRequest.
POST	/salesOrder/{id}/!transform/itemFulfillment	Transform to itemFulfillment.
POST	/salesOrder/{id}/!transform/returnAuthorization	Transform to returnAuthorization.
POST	/salesOrder/{id}/!transform/cashSale	Transform to cashSale.
POST	/salesOrder/{id}/!transform/revenueCommitment	Transform to revenueCommitment.
POST	/salesOrder/{id}/!transform/invoice	Transform to invoice.
POST	/salesOrder/{id}/!transform/storePickupFulfillment	Transform to storePickupFulfillment.

**Note:** Links to NetSuite records that are out of the scope of the select parameter are declared as generic JSON objects with no structure in the returned record metadata. You can avoid this by listing the referenced record in the select parameter.

From the graphic representation, you can see how to obtain a list of all customer and sales order records, how to create new record instances, and how to perform read, update, and delete operations upon them. You can expand each REST method for more description. The Swagger Editor output also contains a **Schema** section. The figure below shows how to use it to explore the structure of a record. For instance, you can see that the salesorder record contains the item sublist, represented in the form of salesorder-itemCollection that contains the **totalResults**, links, and items properties. The structure of a single line of the item sublist is then captured in the salesorder-itemElement part.



```

salesOrder-itemCollection ▾ {
  description:          This record is available as a beta record.

  links                Links ▾ [
    title: Links
    readOnly: true

    nsLink > {...}]
  totalResults        integer($int64)
    title: Total Results
    readOnly: true
  count               integer($int64)
    title: Count
    readOnly: true
  hasMore             boolean
    title: Has More Results
    readOnly: true
  offset              integer($int64)
    title: Query Offset
    readOnly: true
  items               ▾ [salesOrder-itemElement ▾ {
    description:          This record is available as a beta record.

    links                Links > [...]
    rateSchedule         string
      nullable: true
    custool_iodate       string($date)
      title: Item Options Date
      nullable: true
      x-ns-custom-field: true
    quantityFulfilled   number($float)
      title: <NULL>
      nullable: true
    oldItemId            string
      nullable: true
    altId                integer($int64)
      nullable: true
    shipCarrier          string
      title: Carrier
      nullable: true
    Enum:
      > Array [ 2 ]
    refAmt               number($double)
      nullable: true
    revrecterminmonths  integer($int64)
      nullable: true
    commitInventory      string
      title: Commit
      nullable: true
    Enum:
      > Array [ 3 ]
  ]
}

```

## Working with JSON Schema Metadata

Similarly to OpenAPI 3.0 metadata, you can obtain the JSON Schema description by setting the value of the Accept HTTP header to **application: schema+json**, as in the following example.

```

SET-HEADER Accept: application/schema+json
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/customer

```

The following is a shortened example of a response to the previous request. Note that the metadata provided in JSON Schema format only describes the internal structure of a resource, for example, its fields, sublists, or subrecords, and links to related resources, and does not describe URLs, HTTP methods, and parameters.

```

{
  "type": "object",
  "properties": {
    "custentity_integer": {
      "title": "Integer Number",
      "type": "integer",
      "format": "int64",
      "nullable": true,
      "x-ns-custom-field": true
    },
    "startDate": {
      "title": "Start Date",
      "type": "string",
      "description": "<p>Enter the date this person or company became a customer, lead or prospect.</p><p>If this person or company has a contract with you, enter the start date of the contract.</p><p>If you enter an estimate or an opportunity for this customer, this field will be updated with the date of that transaction.</p>",
      "format": "date",
      "nullable": true
    },
  },
}

```

"\$schema": "https://json-schema.org/draft-06/hyper-schema#"

**Note:** The JSON generated by the beta version of the REST API generates external URL links leading to metadata about the linked record types. These are, however, reachable only locally and are unreachable for external validators (such as the JSON Schema validator).

The JSON Schema record metadata received in the previous step can be used as input for any JSON Schema Draft 6 compatible library and tool for further processing. The most frequent use of the JSON Schema record metadata is input and output validation. The following JSON object is an example representing a customer record.

```

{
  entityid: "My Customer",
  currency: { "id": "1" },
  representingsubsidiary: { "id": "1" },
  monthlyclosing: "31 - End of the Month"
}

```

You can check if this record representation conforms to the record metadata schema by using the freely available [JSON Schema Validator](#).

### JSON Schema Validator

newtonsoft.com

An online, interactive JSON Schema validator. Supports JSON Schema Draft 3, Draft 4, Draft 6 and Draft 7. [View source code](#)

Select schema:

```
1 {
2   "required": [
3     "currency",
4     "entityid",
5     "monthlyclosing"
6   ],
7   "type": "object",
8   "properties": {
9     "links": {
10      "title": "links",
11      "type": "array",
12      "readOnly": true,
13      "items": {
14        "type": "object",
15      }
16    },
17    "parent": {
18      "title": "Parent",
19      "type": "object",
20    },
21    "referrerlist": {
22      "title": "Referrer",
23      "type": "object",
24    },
25    "startdate": {
26      "title": "Start Date",
27      "type": "string",
28      "format": "date"
29    }
30  }
31 }
```

Input JSON:

```
1 {
2   entityid: "My Customer",
3   currency: { "id": "1" },
4   monthlyclosing: "31 - End of the Month"
5 }
```

✔ No errors found. JSON validates against the schema

# Working with Records

View the [Using REST Web Services for Custom Record Operations](#) video.

Using REST web services, you can perform CRUD (create, read, update, delete) operations on NetSuite records. The following sections provide information about the structure of NetSuite records, and the ways you can work with records using REST web services.

- [The REST API Browser](#)
- [NetSuite Record Structure](#)
- [Using CRUD Operations on Custom Records](#)
- [Creating a Record Instance](#)
- [Getting a Record Instance](#)
- [Updating a Record Instance](#)
- [Using the Upsert Operation](#)
- [Deleting a Record Instance](#)
- [Using External IDs](#)
- [Using Datetime Fields](#)
- [Executing Record Actions](#)
- [Transforming Records](#)
- [Using the REST Web Services SuiteScript Execution Context](#)

## The REST API Browser

Go to the [REST API Browser](#).

The REST API Browser is a browser that provides a visual overview of the structure and capabilities of the REST web services Record API. The data presented in the REST API Browser is based on OpenAPI 3.0 metadata. For information about metadata, see [Working with OpenAPI 3.0 Metadata](#).

The REST API browser provides the following information:

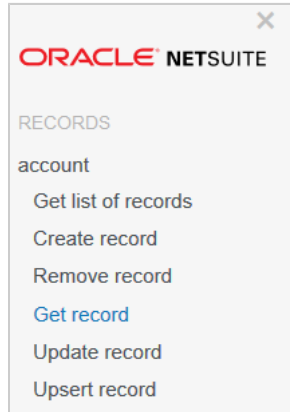
- The support level of records. Beta records are marked with a beta label.
- The summary of all records, sublists, schema definitions, and other objects.
- The available operations you can perform on a record.
- The description of URLs, HTTP methods, and request parameters used for CRUD operations.
- The structure of responses you can receive after performing an operation.
- The structure of error messages.
- The description of field names and field types, and the properties of fields.
- The subformat of strings associated with specialized fields such as date and time fields.

## Navigating in the REST API Browser

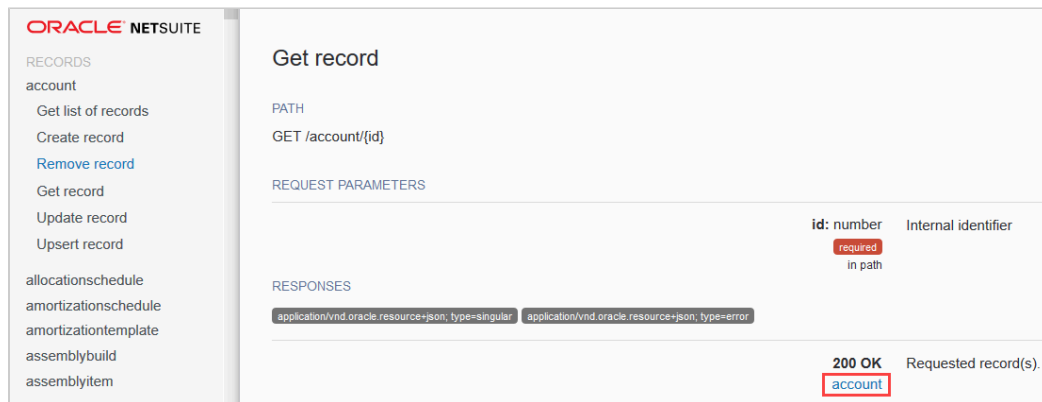
The REST API Browser is designed to let you browse for records and schema definitions in an alphabetical order.

## To view records and schema definitions:

1. Click the name of the appropriate record in the left pane.
2. Click the operation you want to view.



3. Review the path, the request parameters, and the responses.
4. Click the linked resource in the Responses section. By clicking the link, you can navigate to the schema definitions section of the REST API Browser.



5. In the schema definitions section, review the properties of the resource.

SCHEMA DEFINITIONS

- account
- account-localizationsColle...
- account-localizationsElem...
- accountCollection
- accountingbook
- accountingbook-accountin...
- accountingbook-accountin...
- accountingbookCollection
- accountingcontext
- accountingcontextCollection
- accountingperiod
- accountingperiod-fiscalCal...
- accountingperiod-fiscalCal...
- accountingperiodCollection
- advintercompanyjournalen...
- advintercompanyjournalen...
- advintercompanyjournalen...
- advintercompanyjournalen...
- allocationschedule
- allocationschedule-allocati...
- allocationschedule-allocati...
- allocationschedule-allocati...

## account

FIELDS

- acctname** *Name*: string
- acctnumber** *Number*: string
- billableexpensesacct**: [account](#)
- category1099misc**: [x-ns-generic](#)
- class**: [classification](#)
- currency**: [currency](#)
- deferralacct**: [account](#)
- department**: [department](#)
- description** *Description*: string
- eliminate** *Eliminate Intercompany Transactions*: boolean
- externalld** *External ID*: string
- id** *Internal ID*: string
- includechildren** *Include Children*: boolean
- inventory** *Inventory*: boolean
- isinactive** *Inactive*: boolean
- legalname** *Legal Name*: string
- links** *links*: [x-ns-link](#) read only

ITEMS

- [x-ns-link](#)

## NetSuite Record Structure

The figure below shows a standard customer record in NetSuite.

The screenshot shows a NetSuite Customer record for Alan Rath. The record is displayed in a tabbed interface with the 'Address' tab selected. The main area shows customer details like name, ID, and email. A red box highlights the main body fields (1) and the address subtab (3). A green circle with the number 2 highlights the 'DEFAULT SHIPPING' field in the subtab.

The figure outlines the basic components of the record:

1. Body fields - for example, name, ID, or email. Body fields are placed either on the main area of the record or on a subtab.

2. Sublists - for example, the Info sublist on a customer record. A sublist consists of line items and their fields. For more information, see the help topic [What is a Sublist?](#)
3. Subrecords - for example, the address subrecord. A subrecord includes many of the same elements as standard NetSuite records, for example, body fields, sublists, and sublist fields. However, you can only create, edit, remove, or view a subrecord from within the context of its parent record. For more information, see the help topic [What is a Subrecord?](#)

You can use REST web services to get and set values on body fields and sublists. You can also use REST web services to create these components.

Creating, updating, and deleting subrecords is not supported in the beta version.

## Using CRUD Operations on Custom Records

REST web services enables you to perform CRUD (create, read, update, and delete) operations on custom record instances. You can use CRUD operations to perform business processing on custom records and move dynamically between custom records.

In NetSuite, you track all of the information in your account using records. Users with the Custom Record Entries permission can create custom records that collect information specific to the needs of their business.

### Usage Notes

Consider the following information when you perform CRUD operations on custom records.

- Creating custom record types through REST web services is not supported. To work with custom record definitions, you must have the Custom Records feature enabled on your account. To enable the feature, go to Setup > Company > Setup Tasks > Enable Features, and select Custom Records on the SuiteCloud subtab. You must create a custom record type in the UI, and then you can work with instances of the custom record type through REST web services. For information, see the help topic [Creating Custom Record Types](#). You also can use SDF to create custom record types. For information, see the help topic [Custom Record Types as XML Definitions](#).
- The record owner of a custom record can be changed through REST web services even if the Allow change option is not selected on the custom record.
- The Last Modified field of a custom record is not accessible through REST web services.
- The Show ID field is always displayed through REST web services, even if the field value is not set in the UI.

For information about using CRUD operations, see the following help topics.

- [Creating a Record Instance](#)
- [Getting a Record Instance](#)
- [Updating a Record Instance](#)
- [Deleting a Record Instance](#)

## Creating a Record Instance

In REST web services, you can create a new record using the POST HTTP method. The POST method expects a request body (a JSON object) that conforms to the record's metadata schema and contains values for at least each mandatory field of the given record type.

The REST beta version does not support the update of subrecords.

**Note:** Not all record fields can be set using the POST method. For example, the id field is read-only.

Any omitted fields are considered empty or to have default values. The following example shows how to create a new customer record instance.

```
POST https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer
{ "entityid": "New Customer", "companyname": "My Company", "subsidiary": { "id": "1" } }
```

The following is an excerpt from the response headers of a successful operation, with no body content returned (HTTP Code 204). The URL of the newly created record is given in the Location header of the response.

```
Date Fri, 04 Jan 2019 08:50:20 GMT
Location https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/647
Content-Type application/json
```

## Getting a Record Instance

In the following example, an instance of the customer record type created in the previous section is retrieved. You can get a particular customer instance by sending a request in the following form: /services/rest/record/v1/customer/<id>. In the following example, the id of the newly created record is 107. You can read the record instance using the following request.

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107
```

The following is an excerpt from the response headers of a successful operation, returned with HTTP Code 200.

```
Date Thu, 14 Feb 2019 14:05:27 GMT
Content-Length 6246
Content-Type application/vnd.oracle.resource+json; type=singular
```

The following is a shortened example of the body of the response.

```
{
  "links": [
    {
      "rel": "self",
      "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107"
    }
  ],
  "accessRole": {
    "links": [
      {
        "rel": "self",
        "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/"
      }
    ]
  },
}
```



```

    "id": "14",
    "refName": "Customer Center"
  },
  "accessTabChanged": false,
  "addressbook": {
    "links": [
      {
        "rel": "self",
        "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107/
addressbook"
      }
    ]
  },
  "autoName": true,
  "balance": 0,
  "billPay": false,
  "bulkmerge": {
    "links": [
      {
        "rel": "self",
        "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107/
bulkmerge"
      }
    ]
  },
},

```

The response is in a format that conforms to the Swagger and JSON schema. You can validate the retrieved data against the JSON Schema. For information about validation, see [Working with JSON Schema Metadata](#).

In the response, note the Hypermedia As The Engine Of Application State (HATEOAS) links elements. Using the links elements, you can navigate through the REST endpoint.

## Format of Sublists and Subrecords

The REST web services beta version does not automatically expand sublists and subrecords. You can use the `expandSubResources` query parameter to expand sublists and subrecords. If the query parameter is not used, the response contains only the body fields of the record, and the sublists and subrecords are represented by links. See the example in [Getting a Record Instance](#).

The following is an example where the query parameter is used.

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107?expandSubResources=true
```

In the response, note the expansion of the addressbookaddress subrecord.

```

"accessTabChanged": false,
  "addressbook": {
    "links": [
      {
        "rel": "self",
        "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107/address
book"
      }
    ]
  }

```

```

    ],
    "items": [
      {
        "links": [
          {
            "rel": "self",
            "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107/
addressbook/39"
          }
        ],
        "addressbookaddress": {
          "links": [
            {
              "rel": "self",
              "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/custom
er/107/addressbook/39/addressbookaddress"
            }
          ],
          "addr1": "417 Washington Blvd",
          "addressee": "Glenrock General Hospital",
          "addrText": "Alan Rath\nGlenrock General Hospital\n417 Washington Blvd\nGlenrock WY 82637",
          "attention": "Alan Rath",
          "city": "Glenrock",
          "country": "US",
          "dropdownstate": {
            "links": [
              {
                "rel": "self",
                "href": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/"
              }
            ],
            "id": "WY",
            "refName": "Wyoming"
          },
          "id": "152",
          "nKey": "152.0",

```

## Format of Selects and References

References to records in the REST web services beta version contain an internal ID (id), a reference name (refName) (that is, the value obtained by invoking the `getFieldText()` method on the reference field), and an HATEOAS link navigating to the referenced record.

```

"currency": {
  "links": [
    {
      "rel": "self",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/currency/1"
    }
  ],
  "refName": "USD",
  "id": "1"
}

```

## Format of Multiselects

The REST web services beta version models multiselects as a collection of references containing the internal ID and HATEOAS links navigating to the referenced record. The following example shows the subsidiary multiselect field that is located on the account record.

```

"subsidiary": {
  "links": [
    {
      "rel": "self",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/account/106/subsidiary"
    }
  ],
  "items": [
    {
      "links": [
        {
          "rel": "self",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/subsidiary/1"
        }
      ],
      "id": "1"
    },
    {
      "links": [
        {
          "rel": "self",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/subsidiary/2"
        }
      ],
      "id": "2"
    }
  ],
  "totalResults": 2
}


```

## Format of Enumeration Values

The beta version of REST web services uses **internal** values of enumeration values.

## Updating a Record Instance

In the following example, the name of a specific customer record instance is updated. In REST web services, you can perform such an update using the PATCH HTTP method. The PATCH method expects a request body with the same fields that can be retrieved using the GET method. That is, the record's metadata schema is shared between reads and updates.

 **Note:** Not all record fields can be updated using the PATCH method. For example, the id field is considered to be read-only.

Any omitted fields are considered unchanged.

In this example, the name (entityid) of the record instance 107 retrieved in the previous example is changed from "Alan Rath" to "Updated Customer". Send the following request to perform the update.

```
PATCH https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107
BODY { "entityid": "Updated Customer" }
```

The following is an excerpt from the response headers of a successful operation, with no body content returned (HTTP Code 204). The URL of the updated record is given in the Location header of the response.

```
Date Fri, 04 Jan 2019 09:07:01 GMT
Location https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107
Content-Type application/json
```

## Using the Upsert Operation

The upsert operation enables you to either create a record, or update an existing record. You can only use the upsert operation when you use an external ID in the request URL and when you use the PUT HTTP method. For information about using external IDs, see [Using External IDs](#).

You can use the upsert operation as a synchronization tool. When using the upsert operation, you do not need to be concerned whether the record with the given external ID already exists.

The following example shows how to use the upsert operation. If the record does not exist, it will be added. If the record already exists, it will be updated.

```
PUT http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/eid:CID002
{
  "firstName": "John",
  "lastName": "Smith"
}
```

## Deleting a Record Instance

To delete a record instance, you need to specify the record type and the instance identifier. The following is a delete request.

```
DELETE https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107
```

The following is an excerpt from the response headers of a successful operation, with no body content returned (HTTP Code 204).

```
Date Thu, 14 Feb 2019 15:38:21 GMT
Content-Type application/json
```

## Using External IDs

Each record in NetSuite can be uniquely identified by its record type in combination with either an external ID or a system-generated internal ID. For an overview of internal and external IDs, see the help topic [Using Internal IDs, External IDs, and References](#).

You can use an external ID as a key to a record instead of an internal ID. The main use of external IDs is during synchronization with existing data outside of NetSuite.

In REST web services, an external ID starts with the prefix "eid:" in the following format:  
**eid:external\_id**.

An external ID can be any string containing letters, numbers, underscore (\_), and hyphen (-).

You can use external IDs anywhere in the URL where an internal ID can be used. You can also use an external ID in the request body with the field name **externalId**.

The following example adds a record. The external ID is used in the request body.

```
POST http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer
BODY {
  "firstName": "John",
  "lastName": "Smith",
  "isPerson": true,
  "externalId": "CID001",
  "subsidiary": {
    "id": "1"
  }
}
```

The following example retrieves a record. External IDs are used in the request to identify the record.

```
GET http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/eid:CID001/subscriptions/eid:SUID042
```

## Using Datetime Fields

You can work with date and datetime fields using the date and datetime types in NetSuite. See the following sections for information about working with date and datetime types.

When setting a date field using REST web services, you must enter a date or datetime value. You can also specify a time zone in your request. Datetime values are returned in UTC format.

## Types of Date-related Fields

The following four types of date-related fields are used in NetSuite.

Type	Example	Notes
Date fields	10/05/2018	Date fields are used, for example, on custom records.
Time fields	10:15 am	Time fields are used, for example, on custom records.
Datetime fields	10/05/2018 10:15 am	Datetime fields are used, for example, on custom records.
Duration fields	25:45 (hours:minutes)	Duration fields are used in time tracking, for example, on time bill records.

## Date Field

Date fields use the date element of datetime fields. For example: 2017-07-21. Date field values are not subject to any time zone conversion.

## Time Fields

Time fields use the time element of datetime fields, with hours, minutes and second. For example: 17:32:28. Time field values are not subject to any time zone conversion.

## Datetime Fields

REST web services use datetime fields in ISO 8601 format and in the UTC time zone. For example: 2017-07-21T17:32:28Z.

Date is the full-date notation as defined by [RFC 3339, section 5.6](#), for example, 2017-07-21.

Date-time is the date-time notation as defined by [RFC 3339, section 5.6](#), for example, 2017-07-21T17:32:28Z.


All datetime values loaded from NetSuite are converted to the UTC time. REST web services accepts datetime values in the same format and in the UTC time zone.

In REST web services, you can also specify the time zone in your request. For example, 2017-07-21T17:32:28+01:00.

## Duration Fields

Duration fields use the same format as in UI, that is, "hh:mm". For example: "25:42". Duration field values are not subject to any time zone conversion.

## Executing Record Actions

 **Warning:** Record action execution through REST web services is a beta feature. The contents of this feature are preliminary and may be changed or discontinued without prior notice. Any change may impact the feature's operation with the NetSuite application. Warranties and product service levels do not apply to this feature or the impact of the feature on other portions of the NetSuite application. We may review and monitor the performance and use of this feature. The documentation for this feature is also considered a beta version and is subject to revision.

REST web services support APIs that provide the programmatic equivalent of clicking a button in the NetSuite user interface. With the record action APIs, you can use REST web services to trigger the same business logic that is triggered by the click of a UI button. Record actions can increase productivity by automating regular tasks that previously had to be done manually in the UI.

By using record actions, you can update the state of a record. Approve and reject are two examples of record actions. When an approve or reject action is executed on a record, the approval status of the record is saved immediately.

REST web services support the same record actions supported by SuiteScript.

For a list of supported actions, see the help topic [Supported Record Actions](#).

Metadata is not provided for record actions.

You can call record actions using the HTTP POST method on the record instance. The action name is prefixed with the string "@".

Action parameters can be passed in the request body in the JSON object.

The following is an example of a record action call with action parameters in the request body.

```
POST http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/vendorPayment/3/@confirm
{
  "confirmationDate": "2019-1-31",
  "exchangeRate": "4.2",
  "postingPeriod": 348
}
```

You can also execute record actions without providing action parameters, as in the following example.

```
POST http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/vendorPayment/3/@confirm
```


If an action is performed successfully, the HTTP 200 response is returned.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oracle.resource+json; type=singular
{
  "links": [
    "self": "http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/vendorPayment/3/@confirm"
  ],
  "result": true
}
```

When a record action cannot be executed, an HTTP 4xx response is returned.

```
HTTP/1.1 400 Bad Request
Content-Type: application/vnd.oracle.resource; type=error
{
  "type": "https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html",
  "title": "Payment already declined",
  "detail": "A payment you are trying to decline is already declined",
  "status": 400,
  "o:errorCode": "USER_ERROR"
}
```

## Transforming Records

 **Warning:** Record transformation through REST web services is a beta feature. The contents of this feature are preliminary and may be changed or discontinued without prior notice. Any change may impact the feature's operation with the NetSuite application. Warranties and product service levels do not apply to this feature or the impact of the feature on other portions of the NetSuite application. We may review and monitor the performance and use of this feature. The documentation for this feature is also considered a beta version and is subject to revision.

In REST web services, you can transform a record from one type into another, using data from an existing record. For example, you can create an invoice record from an existing sales order record, using data from the sales order.

All record transformations available in SuiteScript are available in REST web services, too. Transformations are also supported in metadata. For information about the supported transformation types, see the help topic [record.transform\(options\)](#).

In REST web services, you use the POST method to send a record transformation request. The record transformation is executed in a single request. In the request URL, the following details must be specified:

- The record type you that you want to transform.
- The ID of the record that you want to transform.
- The transformation action. The transformation action must be formatted in the following way in the URL: `!transform`.
- The record type you that you want transform into.

In the following example, a sales order record is transformed into an invoice. If the transformation is performed successfully, an HTTP 204 – No Content response is returned.

```
POST http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/salesOrder/3!/transform/invoice
```

In the request body, you can also specify data for additional fields of the record, as shown in the following example.

```
{
  "memo": "REST, end date and bills date added",
  "enddate": "2020-01-14",
  "billdate": "2020-01-14",
  "item": {
    "items": [
      {
        "item": { "id": "3456" },
        "amount": 1
      }
    ]
  }
}
```

## Using the REST Web Services SuiteScript Execution Context

Execution contexts provide information about how or when a script or workflow is triggered to execute. For example, a script can be triggered in response to an action in the NetSuite application or an action occurring in another context, such as a web services integration. You can use execution context filtering to ensure that your scripts or workflows are triggered only when necessary. This filtering can improve performance in contexts where the scripts or workflows are not required.

For NetSuite user event and client scripts, you can set up execution context filtering to control the contexts in which these scripts can run. You can set up execution context filtering in the Execution Context field on the Context Filtering subtab of the Script Deployment record. For information about script deployment, see the help topic [Methods of Deploying a Script](#). For more information about execution contexts in general, see the help topics [Execution Contexts](#) and [runtime.executionContext](#).

If a script has its execution context set to REST web services, the script only executes if it is triggered through a REST web services request.

For example, you can deploy a script similar to the following on customer records. In this example, the `beforeSubmit` script is only executed on the record if the request is triggered through REST web services.

```
define(['N/record', 'N/runtime'],
  function(record, runtime) {
    function beforeSubmit(context) {
```



```
log.debug("Before submit of " + context.newRecord.type);  
var executionContext = runtime.executionContext;  
var isRestWs = (runtime.ContextType.REST_WEBSERVICES === executionContext);  
log.debug("Is originating from REST WS: " + isRestWs);  
}  
})
```

After running the script, you can see its execution context in the script execution log. See the help topic [Using the Script Execution Log Tab](#). You can also find information about the execution context in the system notes of a record. See the help topic [Viewing System Notes on a Record](#).

# Working with Sublists

Sublists are an important part of several NetSuite record types. In the REST web services beta, you can use the POST and PATCH operations to access a sublist on a record instance. However, because sublists in NetSuite are of multiple types, for instance, editable sublist, applicable sublist, or search sublist, each sublist can have slightly different behavior. Some sublists on particular records can even contain pre-generated lines. Moreover, if a line is added based on user input, it can contain default or computed values. Some sublists include one or more key fields that can uniquely identify each sublist record. These are keyed sublists. The behavior of keyed sublists is different from the behavior of non-keyed sublists.

**Note:** For information about the format of sublists in REST web services, see [Format of Sublists and Subrecords](#).

The following list includes all operations that can be performed with a sublist:

1. Add new
  - Single line
  - Multiple lines at once
2. Update existing
  - Single line
  - Unset particular field
  - Multiple lines at once
3. Remove existing
  - Single line
  - Multiple lines at once
  - All
4. Mix (any combination of the above at once)
5. Replace all

Operations 2.a, 2.b, 2.c, 3.a, 3.b, and 4 are not possible on non-keyed sublists. You can achieve the equivalent of these operations by replacing all sublist lines.

SOAP web services offer a special **replaceAll** attribute to enable some of these operations. For more information about this topic, see the help topic [Updating Sublists in SOAP Web Services](#).

Similar behavior is also reflected in REST web services. This section provides details about each sublist operation mode.

In all the examples in this section, assume the following initial state of a record instance (unless otherwise specified).

```
GET .../myrecord/100

{
  "body1": "previous body text 1",
  "body2": "previous body text 2",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "previously present line 1" },
      { "key1": "b", "key2": "2", "col": "previously present line 2" },
      { "key1": "X", "key2": "0", "col": "previously present line 0" },
    ]
  }
}
```

```

    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "previously present line 1" },
      { "col": "previously present line 2" },
      { "col": "previously present line 0" },
    ]
  }
}

```

See the following topics for more details:

- [Creating a Sublist](#)
- [Updating a Sublist](#)
- [Replacing a Sublist](#)

## Creating a Sublist

During create operations, lines of a keyed sublist are updated if the line contains matching keys, and the remaining lines from the request are added. For non-keyed sublists, the lines are always appended to the ones that are already present on the sublist by default. The following is an example of a POST operation upon a sublist.

## Request Body

POST .../myrecord

```

{
  "body1": "inserted body text 1",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "inserted line 1" },
      { "key1": "b", "key2": "2", "col": "inserted line 2" }
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" }
    ]
  }
}

```

## Response – Post State

Response:

201 Created

Location: .../myrecord/101

```

{
  "body1": "inserted body text 1",
  "body2": "default body text 2",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "inserted line 1" },
      { "key1": "b", "key2": "2", "col": "inserted line 2" },
      { "key1": "X", "key2": "0", "col": "default line 0" },
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "default line 1" },
      { "col": "default line 2" },
      { "col": "default line 0" },
      { "col": "inserted line 1" },
      { "col": "inserted line 2" },
    ]
  }
}

```

**Note:** Keyed sublist lines are updated if the key matches, otherwise, a new line is created (in other words, sublist lines are merged). Lines for non-keyed sublists are added.

## Updating a Sublist

During update operations, lines of a keyed sublist are updated if the line contains matching keys, and the remaining lines from the request are added. For non-keyed sublists, the lines are always appended to the ones that are already present on the sublist. The following examples show PATCH operations upon a sublist.

## Request Body – Patch Operation

PATCH .../myrecord/100

```

{
  "body1": "replaced body text 1",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "replaced line 1" },
      { "key1": "b", "key2": "2", "col": "replaced line 2" }
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" }
    ]
  }
}

```


## Response – Post State

Response:

204 No Content

Post State:

```
{
  "body1": "replaced body text 1",
  "body2": "previous body text 2",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "replaced line 1" },
      { "key1": "b", "key2": "2", "col": "replaced line 2" },
      { "key1": "X", "key2": "0", "col": "previously present line 0" },
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "previously present line 1" },
      { "col": "previously present line 2" },
      { "col": "previously present line 0" },
      { "col": "inserted line 1" },
      { "col": "inserted line 2" },
    ]
  }
}
```

 **Note:** Keyed sublist lines are updated if the key matches, otherwise, a new line is created (in other words, sublist lines are merged). All lines of non-keyed sublists are added.

## Request Body – Nullifying a Sublist

PATCH .../myrecord/100

```
{
  "body1": "replaced body text 1",
  "sublist" : {
    "items" : null
  }
}
```

```
{
  "body1": "replaced body text 1",
  "sublist" : null
}
```

## Response – Post State

Response:

204 No Content

Post State:

```
{
  "body1": "replaced body text 1",
  "body2": "previous body text 2",
  "sublist" : {
    "items" : []
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "previously present line 1" },
      { "col": "previously present line 2" },
      { "col": "previously present line 0" },
    ]
  }
}
```

**Note:** Keyed as well as non-keyed sublist lines are replaced. If the sublist is mandatory, it is expected that the operation fails.

## Replacing a Sublist

During sublist replacements, lines of a keyed sublist are updated if the line contains matching keys, other lines are removed, and the remaining unmatched lines from the request are added. For non-keyed sublists, all lines are always removed and replaced by lines that are in the incoming request. The following examples show POST and PATCH operations with a replace query parameter.

### Request Body – Replacing a Default Keyed Sublist

POST .../myrecord?replace=sublist

```
{
  "body1": "inserted body text 1",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "inserted line 1" },
      { "key1": "b", "key2": "2", "col": "inserted line 2" }
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" }
    ]
  }
}
```

### Response – Post State

Response:

201 Created

Location: .../myrecord/101

```

{
  "body1": "inserted body text 1",
  "body2": "default body text 2",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "inserted line 1" },
      { "key1": "b", "key2": "2", "col": "inserted line 2" },
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "default line 1" },
      { "col": "default line 2" },
      { "col": "default line 0" },
      { "col": "inserted line 1" },
      { "col": "inserted line 2" },
    ]
  }
}

```

**Note:** Keyed sublist lines are updated if the key matches, other lines are removed, and unmatched lines from the request are created as new. All lines of non-keyed sublists are added. The operation fails if any replaced default sublist line is read-only.

## Request Body – Replacing a Default Non-Keyed Sublist

POST .../myrecord?replace=unkeyedsublist

```

{
  "body1": "inserted body text 1",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "inserted line 1" },
      { "key1": "b", "key2": "2", "col": "inserted line 2" }
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" }
    ]
  }
}

```

## Response – Post State

Response:

201 Created

Location: .../myrecord/101

```
{
  "body1": "inserted body text 1",
  "body2": "default body text 2",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "inserted line 1" },
      { "key1": "b", "key2": "2", "col": "inserted line 2" },
      { "key1": "X", "key2": "0", "col": "default line 0" },
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" },
    ]
  }
}
```

**Note:** Keyed sublist lines are updated if the key matches, otherwise, a new line is created. All lines of non-keyed sublists are removed and then created as new. The operation fails if any replaced default sublist line is read-only.

## Request Body – Replacing a Keyed Sublist

PATCH .../myrecord/100?replace=sublist

```
{
  "body1": "replaced body text 1",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "replaced line 1" },
      { "key1": "b", "key2": "2", "col": "replaced line 2" }
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" }
    ]
  }
}
```

## Response – Post State

Response:

204 No Content

Post State:

```
{
```



```

"body1": "replaced body text 1",
"body2": "previous body text 2",
"sublist" : {
  "items" : [
    { "key1": "a", "key2": "1", "col": "replaced line 1" },
    { "key1": "b", "key2": "2", "col": "replaced line 2" },
  ]
},
"unkeyedsublist" : {
  "items" : [
    { "col": "previously present line 1" },
    { "col": "previously present line 2" },
    { "col": "previously present line 0" },
    { "col": "inserted line 1" },
    { "col": "inserted line 2" },
  ]
}
}

```

**Note:** Keyed sublist lines are updated if the key matches, other lines are removed, and unmatched lines from the request are created as new. All lines of non-keyed sublists are added.

## Request Body – Replacing a Non-Keyed Sublist

PATCH .../myrecord/100?replace=unkeyedsublist

```

{
  "body1": "replaced body text 1",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "replaced line 1" },
      { "key1": "b", "key2": "2", "col": "replaced line 2" }
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" }
    ]
  }
}

```

## Response – Post State

Response:

204 No Content

Post State:

```

{
  "body1": "replaced body text 1",

```

```

"body2": "previous body text 2",
"sublist" : {
  "items" : [
    { "key1": "a", "key2": "1", "col": "replaced line 1" },
    { "key1": "b", "key2": "2", "col": "replaced line 2" },
    { "key1": "X", "key2": "0", "col": "previously present line 0" },
  ]
},
"unkeyedsublist" : {
  "items" : [
    { "col": "inserted line 1" },
    { "col": "inserted line 2" },
  ]
}
}

```

**Note:** Keyed sublist lines are updated if the key matches, otherwise, a new line is created (in other words, sublist lines are merged). All lines of non-keyed sublists are removed and then created as new.

## Request Body – Replacing Multiple Sublists

The following operation is a combination of the previous operations.

PATCH .../myrecord/100?replace=sublist,unkeyedsublist

```

{
  "body1": "replaced body text 1",
  "sublist" : {
    "items" : [
      { "key1": "a", "key2": "1", "col": "replaced line 1" },
      { "key1": "b", "key2": "2", "col": "replaced line 2" }
    ]
  },
  "unkeyedsublist" : {
    "items" : [
      { "col": "inserted line 1" },
      { "col": "inserted line 2" }
    ]
  }
}

```

## Response – Post State

Response:

204 No Content

Post State:

```

{
  "body1": "replaced body text 1",
  "body2": "previous body text 2",

```

```
"sublist" : {  
  "items" : [  
    { "key1": "a", "key2": "1", "col": "replaced line 1" },  
    { "key1": "b", "key2": "2", "col": "replaced line 2" },  
  ]  
},  
"unkeyedsublist" : {  
  "items" : [  
    { "col": "inserted line 1" },  
    { "col": "inserted line 2" },  
  ]  
}  
}
```

# Working with Subrecords

Subrecords represent a way of storing data in NetSuite. Like records, subrecords are classified by type. Some common types of subrecord include address, inventory detail, and order schedule.

A subrecord includes many of the same elements of a standard NetSuite record (body fields, sublists and sublist fields, and so on). However, subrecords must be created, edited, removed, or viewed from within the context of a standard (parent) record.

The purpose of a subrecord is to hold key related data about the parent record. For example, a parent record would be a Serialized Inventory Item record. This record defines a type of item. A subrecord would be an Inventory Detail subrecord. This is a subrecord that contains all data related to where the item might be stored in a warehouse. In this way, the subrecord contains data related to the item, but not data that directly defines the item. Without the parent record, the subrecord would serve no purpose. For more information about subrecords in general, see the help topics [What is a Subrecord?](#) and [Understanding Subrecords](#).

In REST web services, you can use the POST and PATCH operations to access a subrecord on the record instance.



**Note:** For information about the format of subrecords in REST web services, see [Format of Sublists and Subrecords](#).

## Updating a Subrecord

Subrecords are modelled as inner JSON properties in the request and response body. In the following example, a subrecord, `addressbookaddress` is set on the sublist line of the `addressbook` sublist.

```
POST http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer
{
  "entityid": "My Customer",
  "companyname": "My Company",
  "email": "another.customer@company.com",
  "addressbook": {
    "items": [{
      "label": "New York HQ",
      "addressbookaddress": {
        "country": "US",
        "state": "NY",
        "zip": "10001",
        "addressee": "Dwight Schrute"
      }
    }]
  }
}
```

## Getting a Subrecord

The following example shows a GET operation, which is used to access a subrecord.

```
GET http://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/42/addressbook/24/addressbookaddress
```

```
{
  "addressee": "Dwight Schrute",
  "city": "New York",
  "country": "US",
  "dropdownstate": {
    "id": "NY",
    "refName": "New York",
    ...
  },
  "state": "NY",
  "zip": "10001",
  ...
}
```

The structure of the subrecord is also returned on the parent record, on the sublist line, if you set the `expandSubResources` query parameter to `true`.

# Record Filtering and Query

The query operation is used to execute a query on a specific record type based on a set of criteria.

Record query only returns record IDs and HATEOAS links. That is, query results have a form of non-expanded references. Additionally, you can only use body fields in query conditions. Saved queries, multilevel joins, and sublist and subrecord queries are not supported in the beta version.

The beta version of REST web services only supports limited record query. Joins are not supported.

For more information, see the following topics:

- [Listing All Record Instances](#)
- [Record Collection Filtering](#)
- [Executing SuiteQL Queries Through REST Web Services](#)
- [Working with SuiteAnalytics Workbooks in REST Web Services](#)
- [Collection Paging](#)

## Listing All Record Instances

You can obtain the list of all records of a record type by sending an HTTP GET request to `.../services/rest/record/v1/record_type`, as shown in the following image.

The screenshot displays a REST client interface for the 'customer' record type. The interface is divided into several sections:

- Method and Path:** A blue button labeled 'GET' is next to the path `/customer`. Below the path, it says 'Get list of records'.
- Parameters:** A section titled 'Parameters' with a 'Try it out' button. It contains the text 'No parameters'.
- Responses:** A table with columns 'Code', 'Description', and 'Links'.
 

Code	Description	Links
200	Get List of records	No links
- Response Body:** A green-bordered box highlights the response body: `application/vnd.oracle.resource+json; type=collection`.

Listing all instances of a given record type can also be understood as a query over the given record type without any criteria. The following is an example of such a request.

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer
```

The following is a shortened example of a response.

```
{
  "links": [
    {
      "rel": "self",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?
limit=1000&offset=0"
    }
  ],
  "items": [
    {
      "links": [
        {
          "rel": "self",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107"
        }
      ],
      "id": "107"
    },
    {
      "links": [
        {
          "rel": "self",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/41"
        }
      ],
      "id": "41"
    },
    {
      "links": [
        {
          "rel": "self",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/90"
        }
      ],
      "id": "90"
    }
  ],
  "totalResults": 3
}
```

## Record Collection Filtering

You can filter the collection of all record instances by using the **q** query parameter to specify filter conditions. Each condition consists of a field name, an operator, and a value. You can join several conditions using the AND / OR logical operators, and you can use **()** to mark precedence.

The following table contains the list of available query operators with their associated field types.

Field Type	Allowed Filters
None	EMPTY, EMPTY_NOT

Field Type	Allowed Filters
Boolean	IS, IS_NOT
Double, Integer, Float, Number, Duration	ANY_OF, ANY_OF_NOT, BETWEEN, BETWEEN_NOT, EQUAL, EQUAL_NOT, GREATER, GREATER_NOT, GREATER_OR_EQUAL, GREATER_OR_EQUAL_NOT, LESS, LESS_NOT, LESS_OR_EQUAL, LESS_OR_EQUAL_NOT, WITHIN, WITHIN_NOT
String	CONTAIN, CONTAIN_NOT, IS, IS_NOT, START_WITH, START_WITH_NOT, END_WITH, END_WITH_NOT
Date / Time	AFTER, AFTER_NOT, BEFORE, BEFORE_NOT, ON, ON_NOT, ON_OR_AFTER, ON_OR_AFTER_NOT, ON_OR_BEFORE, ON_OR_BEFORE_NOT

Not all operators accept one value. Some operators do not require any value, some operators require two values, and some operators accept any number of values. Consider the following examples:

- Unary operators: The EMPTY and EMPTY\_NOT operators do not accept any values. For example: `?q=companyName EMPTY`
- Ternary operators: The BETWEEN, BETWEEN\_NOT, WITHIN, and WITHIN\_NOT operators accept two values. For example: `?q=id BETWEEN_NOT [1, 42]`
- N-ary operators: The ANY\_OF and ANY\_OF\_NOT operators do accept one or any higher number of values. For example: `?q=id ANY_OF [1, 2, 3, 4, 5]`

You can find the field available for filtering in the metadata. For information about metadata, see [Working with Resource Metadata](#).

The following is an example of a simple query.

**Note:** The spaces in URLs are encoded. The following examples are presented without encoding for clarity.

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?q=email START_WITH barbara
```

The response is a collection of customer record instances where the value in the email field starts with the value of barbara. The result is a collection resource containing links to resources that match query criteria. The response could be similar to the following:

```
{
  "links": [
    {
      "rel": "self",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?
limit=1000&offset=0"
    }
  ],
  "items": [
    {
      "links": [
        {
          "rel": "self",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107"
        }
      ],
      "id": "107"
    }
  ],
}
```



```
],
  "totalResults": 1
}
```

When your condition value contains spaces, you should use quotation marks around the constraint, for instance, `firstname IS "Barbara Allen"`. See the following additional query examples:

- Find customer by company name (string value):  
GET `https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?q=companyname START_WITH "Another Company"`
- Find inactive customers (boolean value):  
GET `https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?q=isinactive IS true`
- Find customers created in 2019 (date value, AND operator):  
GET `https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?q=dateCreated ON_OR_AFTER "1/1/2019" AND dateCreated BEFORE "1/1/2020"`
- Find customers with high or low credit limit (number constraint, OR operator):  
GET `https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?q=creditlimit GREATER_OR_EQUAL 1000 OR creditlimit LESS_OR_EQUAL 10`

**Note:** When you join more than one conditions, the AND logical operator has priority over the OR logical operator. To change this behavior, use parentheses as follows: `" ( " and " ) "`.

## Executing SuiteQL Queries Through REST Web Services

**Warning:** SuiteQL query execution through REST web services is a beta feature. The contents of this feature are preliminary and may be changed or discontinued without prior notice. Any change may impact the feature's operation with the NetSuite application. Warranties and product service levels do not apply to this feature or the impact of the feature on other portions of the NetSuite application. We may review and monitor the performance and use of this feature. The documentation for this feature is also considered a beta version and is subject to revision.

SuiteQL is a query language based on the SQL database query language. SuiteQL provides advanced dynamic query capabilities that can be used to access NetSuite records.

For more information about SuiteQL in general, see the help topic [SuiteQL](#).

You can execute SuiteQL queries through REST web services by sending a POST request to the `suiteql` resource, and specifying the query in the request body after the body parameter `q`.

In the request URL you can also specify the number of results you want to return in a single page and the page offset. For information about paging and offset values, see [Collection Paging](#).

The following example shows a SuiteQL query executed through REST web services. Note that `Prefer: transient` is a required header parameter.

```
POST https://demo123.suitetalk.api.netsuite.com/services/rest/query/v1/suiteql?limit=10&offset=10
Prefer: transient
{
```

```

    "q": "SELECT email, COUNT(*) as count FROM transaction GROUP BY email"
  }

```

The following is a shortened response.

```

{
  "links": ...,
  "count": 3,
  "offset": 10,
  "totalResults": 53,
  "items": [
    {
      "links": [],
      "email": "test@netsuite.com",
      "count": "143"
    },
    {
      "links": [],
      "email": "test2@netsuite.com",
      "count": "144"
    },
    {
      "links": [],
      "email": "test3@netsuite.com",
      "count": "145"
    }
  ],
  ...
}

```

In SuiteQL queries, you can also use advanced SQL functions, for example joins and concatenations. The following example shows a request where the first and last names from employee records are concatenated as full name.

```

{
  "q": "SELECT CONCAT(firstname,lastname) as fullname FROM employee"
}

```

## Working with SuiteAnalytics Workbooks in REST Web Services



**Warning:** Support for SuiteAnalytics workbook execution through REST web services is a beta feature. The contents of this feature are preliminary and may be changed or discontinued without prior notice. Any change may impact the feature's operation with the NetSuite application. Warranties and product service levels do not apply to this feature or the impact of the feature on other portions of the NetSuite application. We may review and monitor the performance and use of this feature. The documentation for this feature is also considered a beta version and is subject to revision.

SuiteAnalytics Workbook is an analytical tool available in NetSuite. With SuiteAnalytics Workbook, you can create highly customizable workbooks that combine queries, pivot tables, and charts using a single tool

that leverages a new data source. For information about working with SuiteAnalytics workbooks in the UI, see the help topic [SuiteAnalytics Workbook Overview](#).

In REST web services, you can run saved workbooks and retrieve their results. You can execute both standard and custom workbooks.

When working with SuiteAnalytics workbooks in REST web services, consider the following:

- You must have the SuiteAnalytics Workbook feature enabled. See [REST Web Services Prerequisites and Setup](#).
- The SuiteAnalytics Workbook permission must be assigned to the user's role. See [REST Web Services Prerequisites and Setup](#).
- Workbooks cannot be created or filtered or edited through REST web services.
- Metadata is not provided for saved workbooks.
- The response from a saved analytics execution returned through REST web services may have different format and values than responses for record instance requests.
- Workbooks can have multiple datasets. A REST web services request executes a single data set if the workbook contains only one dataset. You cannot execute workbooks that contain multiple datasets through REST web services.

The following is an example of the execution of a standard workbook.

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/query/v1/workbook/SalesExampleWorkbook/result
```

The following is a response:

```
{
  "links": [
    {
      "rel": "self",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/query/v1/workbook/SalesExampleWorkbook/result?limit=1000&offset=0"
    }
  ],
  "items": [
    {
      "links": [],
      "amount": "5.33333334",
      "amount2": "2",
      "entity": "Customer AU",
      "item": "Item Tax 2 Lbs(AU)",
      "itemcount": "1",
      "postingperiod": "May 2010",
      "trandate": "5/28/2010"
    }
  ],
  "count": 1,
  "hasMore": false,
  "offset": 0,
  "totalResults": 1
}
```

You can set up your own custom workbook in the UI, either by creating a new workbook or by editing a standard workbook. For information about creating a custom workbook, see the help topic [Custom Workbooks and Datasets](#).

You must save the workbook before you can run it, and you must also share it with REST web services users. For information about sharing a workbook with other users, see the help topic [Sharing Workbooks and Datasets](#).

When your custom workbook is ready, you can run it in REST web services by creating a request that contains the workbook identifier. Every custom workbook is identified by the prefix "custworkbook" and contains a workbookId. You can see the workbook ID in the URL in the UI when you edit the workbook. For example: `/report/report.n1?workbookId=2`

The following is an example of executing a custom workbook.

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/query/v1/workbook/custworkbook2/result
```

The following is the response. The response is in a key-value structure or table.

```
{
  "links": [
    {
      "rel": "self",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/query/v1/workbook/custworkbook2/result?limit=1000&offset=0"
    }
  ],
  "items": [
    {
      "links": [],
      "amount": "5.33333334",
      "amount2": "2",
      "entity": "Customer AU",
      "item": "Item Tax 2 Lbs(AU)",
      "itemcount": "1",
      "postingperiod": "May 2010",
      "trandate": "5/28/2010"
    }
  ],
  "count": 1,
  "hasMore": false,
  "offset": 0,
  "totalResults": 1
}
```

## Collection Paging

Lists of record instances and the results of saved analytics workbook searches are returned in one or more pages. The results are displayed on multiple pages, with the default setting of 1000 results per page. You can also specify your own paging by adding the limit criteria to your request.

The following is an example of a request for all instances of customer records, displaying two results per page:

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?limit=2
```

The following is an example of a response:

```
{
```

```

"links": [
  {
    "rel": "next",
    "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/account?limit=2&offset=2"
  },
  {
    "rel": "last",
    "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/account?
limit=22&offset=2"
  },
  {
    "rel": "self",
    "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?
limit=2&offset=0"
  }
],
"items": [
  {
    "links": [
      {
        "rel": "self",
        "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/107"
      }
    ],
    "id": "107"
  },
  {
    "links": [
      {
        "rel": "self",
        "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/41"
      }
    ],
    "id": "41"
  }
],
"count": 2,
"offset": 0,
"hasMore": true,
"totalResults": 3
}

```

The values of the **count**, **hasMore**, and **totalResults** fields provide information about the results that can be retrieved in subsequent requests for other pages. Links to the last and next pages provide direct links that you can use to retrieve these pages.

You can retrieve the next page by sending a request with the offset value specified. The offset provides the index of the result from which you should start the requested page.



**Note:** You can only set offset and limit values in a way that the offset is divisible by the limit. For example, Offset=20, Limit=10 or Offset=0, Limit=5

The following is an example of a request for the second (and last) page of results with all customers:

```
GET https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/account?limit=2&offset=2
```

The following is an example of a response:

```
{
  "links": [
    {
      "rel": "prev",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?
limit=2&offset=0"
    },
    {
      "rel": "first",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?
limit=22&offset=0"
    },
    {
      "rel": "self",
      "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer?
limit=2&offset=2"
    }
  ],
  "items": [
    {
      "links": [
        {
          "rel": "self",
          "href": "https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/customer/90"
        }
      ],
      "id": "90"
    }
  ],
  "count": 1,
  "offset": 2,
  "hasMore": false,
  "totalResults": 3
}
```

# Error Handling and Logging in REST Web Services

For information about error handling and logging, see the following topics:

- [Error Handling in REST Web Services](#)
- [Using the REST Web Services Execution Log](#)

## Error Handling in REST Web Services

In REST web services, HTTP status codes are used to inform you about the success or failure of a request. The following status codes are used.

- 2xx status codes are used for successful requests.
- 4xx status codes are used for failures due to user error.
- 5xx status codes are used for failures due to system error.

Error messages include a title describing the issue, the HTTP status code, and an error code.

The following examples show some common errors with their descriptions.

### Missing Login Information

The following error is returned if the request does not contain the login authorization header.

```
{
  "type": "https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html",
  "title": "Missing login header!",
  "status": 400,
  "o:errorCode": "USER_ERROR"
}
```

### Request for Non-existent Record

The request GET `https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/record/customer/999` returns the following error, because the request is for a non-existent customer.

```
{
  "type": "https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html",
  "title": "Invalid record instance identifier (i.e., id, external id, or name id) abc in request URL",
  "status": 400,
  "o:errorCode": "USER_ERROR"
}
```

### Invalid Request

The request GET `https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/record/customer, salesorder` returns the following error,

because there is an extra space in the query parameters. The correct request would be `https://demo123.suitetalk.api.netsuite.com/services/rest/record/v1/metadata-catalog/record/customer,salesorder`.

```
{
  "type": "https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html",
  "title": "The request could not be understood by the server due to malformed syntax.",
  "status": 400,
  "o:errorCode": "INVALID_REQUEST"
}
```

## Exceeded Concurrency Governance Limit

The following error is returned if the request is rejected due to exceeding the limit allowed by concurrency governance. For information about request limits, see [Concurrency Governance](#).

If this error occurs, you should retry sending the request. For information about implementing a retry logic in your code, see the help topic [Retrying Failed Web Services Requests](#).

```
{
  "type": "https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html",
  "title": "Concurrent request limit exceeded. Request blocked.",
  "status": 429,
  "o:errorCode": "USER_ERROR"
}
```

## System Error

The following error is returned if a system error occurs while the request is being processed. If a system error occurs, contact NetSuite Customer Support and refer to the error ID.

```
{
  "type": "https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html",
  "title": "An unexpected error occurred. Error ID: jrgbpyylphhishmmlxyt",
  "status": 500,
  "o:errorCode": "UNEXPECTED_ERROR"
}
```

## Using the REST Web Services Execution Log

Each integration record includes a subtab labeled REST Web Services, under the Execution Log subtab. This log lists REST web services requests that are uniquely identified with the integration record you are currently viewing. That is, the log includes those requests that referenced the integration record's consumer key.

The execution log includes information about all requests and responses.

To see the execution log for a REST web services integration:

1. Go to Setup > Integration > Manage Integrations.
2. Select an integration record from the list.



3. On the Execution Log subtab, select REST Web Services.

For each request, the log includes the following details:

- The date and time that the request was made.
- The duration of the request.
- The status of the request.
- The email address of the user who sent the request.
- The HTTP method used and the HTTP status code
- The URL to the record or record type.
- The request and response body.

**Note:** The requests and responses logged in the execution log contain the values of sensitive fields in masked out format. For information about using encrypted format on custom fields, see the help topic [Encrypted Custom Field Stored Values](#).

Integration									
APPLICATION ID		STATE	CREATED						
NAME		NOTE	CREATED BY						
DESCRIPTION		CONCURRENCY LIMIT	LAST STATE CHANGE						
		MAX CONCURRENCY LIMIT	LAST STATE CHANGED BY						
		104							
<a href="#">Authentication</a> <a href="#">Execution Log</a> <a href="#">System Notes</a>									
<a href="#">SQAP Web Services</a> <a href="#">REST Web Services</a> <a href="#">RESTJets</a>									
TIME	DURATION (SECS)	STATUS	USER EMAIL	HTTP METHOD	HTTP STATUS CODE	URL	REQUEST	RESPONSE	
11/28/2019 8:06 am	0.097	FAILED	[REDACTED]	POST	400	/services/rest/query/v1/suiteq?limit=10&offset=20	view	view	
11/28/2019 8:02 am	1.049	FAILED	[REDACTED]	POST	400	/services/rest/query/v1/suiteq?limit=10&offset=20	view	view	
11/28/2019 7:38 am	6.343	FINISHED	[REDACTED]	POST	204	/services/rest/record/v1/customer	view	view	
11/28/2019 7:38 am	0.631	FAILED	[REDACTED]	GET	404	/services/rest/query/v1/workbook/AnalyticWorkbookAPITestRecord1/result	view	view	
11/28/2019 7:38 am	1.752	FINISHED	[REDACTED]	OPTIONS	204	/services/rest/*	view	view	
11/25/2019 8:20 am	1.444	FAILED	[REDACTED]	POST	400	/services/rest/query/v1/suiteq?limit=10&offset=20	view	view	
11/25/2019 8:17 am	33.997	FINISHED	[REDACTED]	POST	204	/services/rest/record/v1/customer	view	view	
11/25/2019 8:16 am	5.187	FAILED	[REDACTED]	POST	403	/services/rest/record/v1/customer	view	view	