

Tokyo API Reference

Last updated: July 15, 2023

Some examples and graphics depicted herein are provided for illustration only. No real association or connection to ServiceNow products or services is intended or should be inferred.

This PDF was created from content on docs.servicenow.com. The web site is updated frequently. For the most current ServiceNow product documentation, go to docs.servicenow.com.

Company Headquarters

2225 Lawson Lane
Santa Clara, CA 95054
United States
(408)501-8550

Integrate with ServiceNow

HTTP-based web services allow diverse applications to talk to each other. ServiceNow supports both inbound (provider) and outbound (consumer) web services.

Inbound web services

Inbound web services allow you to access and modify ServiceNow data using a client application.

- REST APIs
- Scripted REST APIs
- Query record data using the GraphQL API framework
- SOAP web service
- JSONv2 web service
- RSS web service
- Scripted SOAP web services

Outbound web services

Outbound web services allow you to send SOAP and REST messages to external web service providers.

- Outbound SOAP web service
- Outbound REST web service
- Inbound web services

You can use any of several web services to integrate with ServiceNow applications, including REST, SOAP, JSONv2, and RSS.

- Outbound web services

You can use any of several outbound web services to integrate with ServiceNow applications, including REST and SOAP.

- [Additional integration resources](#)

In addition to employing inbound and outbound web services to integrate with ServiceNow apps, other resources are available to you, such as scripted GraphQL, Javascript, and API developer guides.

Inbound web services

You can use any of several web services to integrate with ServiceNow applications, including REST, SOAP, JSONv2, and RSS.

- [REST APIs](#)

REST (REpresentational State Transfer) is a simple stateless architecture that provides standards between computer systems on the web, making it easier for them to communicate with each other.

- [SOAP web service](#)

Simple Object Access Protocol (SOAP) is an XML-based protocol for accessing web services over HTTP.

- [JSONv2 web service](#)

The ability to describe sets of data in JSON format is a natural extension to the JavaScript language.

- [RSS web service](#)

RSS (Rich Site Summary) is a format for delivering web-based information that changes regularly.

- [Exporting and converting records into complex data types](#)

Use URL parameters to export table records and convert them into complex data types, such as JSON, XML, PDF, CSV, and XLS.

- [Inbound web service examples](#)

Inbound web service examples demonstrate how to access ServiceNow web services.

REST APIs

REST (REpresentational State Transfer) is a simple stateless architecture that provides standards between computer systems on the web, making it easier for them to communicate with each other.

The Now Platform provides various REST APIs, which are active by default. These APIs provide the ability to interact with various ServiceNow functionality within your application. Such functionality includes the ability to perform create, read, update, and delete (CRUD) operations on existing tables (Table API), insert data into, retrieve information from, and run transforms against a MetricBase database (MetricBase Time Series API), and many others.

For a list of available REST APIs, see [REST API reference](#).

Note: You can view inbound API transactions in the Transaction logs. Use a link like the one below to view the transactions for the current day:

```
https://<instancename>.service-now.com/
syslog_transaction_list.do?
sysparm_query=sys_created_onONToday%40javascript%3Ags.beginningOfToday()
%40javascript%3Ags.endOfToday()%5Etype%3Drest
```

REST URI format and available parameters

ServiceNow REST APIs follow standard REST API protocol. They also provide "custom" URI and query parameters to ensure backwards compatibility and provide additional functionality such as paginating long lists of results. The following sections describe the functionality behind these custom parameters, which are all optional.

REST API versioning

ServiceNow REST API URLs may include a version number, such as /api/now/v1/table/{tableName}. Version numbers identify the endpoint version that a URL accesses. By specifying a version number in your URLs, you can ensure that future updates to the REST API do not negatively impact your integration.

URIs that do not specify a version number in the URI, such as /api/now/table/{tableName}, use the latest REST endpoint for your instance version.

Supported HTTP request methods

- GET
- DELETE
- HEAD
- PATCH
- POST
- PUT

For details on these methods, refer to the [RFC-2616 Hypertext Transfer Protocol](#) document.

Note:

- You can use the HEAD methods in place of GET methods to return a response without a response body.
- You cannot pass multiple records in POST, PUT, and PATCH operations. If you do, only the first record is processed, the rest are ignored.
- You cannot use POST, PUT, and PATCH to insert or update records into a Database view, as Database views are read-only.

Data format headers

REST APIs require the Accept and Content-Type request headers for proper data formatting for requests that contain a request body or response body. POST, PUT, PATCH, and DELETE operations require you to provide both headers. GET and HEAD operations require only the Accept header. Failing to provide the required headers results in a 400 Bad Request error.

For most ServiceNow REST APIs these request headers support the following values:

- Accept: application/json, application/xml
- Content-Type: application/json, application/xml

For the list of specific values supported by each endpoint, refer to the [REST API reference](#).

Other headers

All requests may contain an authentication header that specifies the user credentials to use for authentication.

You can also override HTTP methods, such as GET or POST, by setting the X-HTTP-Method-Override header.

Special data handling

The following describes some of the data handling nuances within the REST API.

- How to clear a data field: Except for choice fields, you can pass an empty value in the parameter to clear the value in the database. For example, sending `{"short_description": ""}` clears the short_description field for the specified record.
- How currency fields are handled: Returned currency values are converted to the local currency based on the user's locale. When inserting data, no conversion is performed. This behavior applies to fields of the types `Currency` or `Price`.

For example, if a user in the UK locale queries records with currency values in USD, the returned values are converted to GBP. However, if this user adds a new record with the currency field value in GBP, the value is stored in GBP without being converted to USD. This GBP value appears in USD if queried by a user in the US locale.

- UI data display versus values passed in a REST endpoint: The UI shows the database `display value`, which is manipulated data. A REST endpoint, by default, inserts and updates the actual values, which can be different from the display value. You can force a REST endpoint to treat passed values as display values by setting the `sysparm_input_display_value` request parameter to true.

Custom query parameters

The ServiceNow REST APIs use the following query parameters across many of the available APIs, providing consistent behavior across the APIs. Use these parameters to paginate large record sets, filter results, and restrict the number of records returned in a single query.

sysparm_limit	<p>Maximum number of records to return. For requests that exceed this number of records, use the sysparm_offset parameter to paginate record retrieval.</p> <p>This limit is applied before ACL evaluation. If no records return, including records you have access to, rearrange the record order so records you have access to return first.</p> <p>Note: Unusually large sysparm_limit values can impact system performance.</p> <p>Data type: Number</p> <p>Default: 1000</p>
sysparm_fields	<p>Comma-separated list of fields to return in the response.</p> <p>Data type: String</p>
sysparm_input_d isplay_value	<p>Flag that indicates whether to set field values using the display value or the actual value. Depending on the different types of fields, the endpoint may manipulate the passed in display values to store the proper values in the database. For example, if you send the display name for a reference field, the endpoint stores the sys_id for that value in the database. For date and time fields, when this parameter is true, the date and time value is</p>

	<p>adjusted for the current user's timezone. When false, the date and time value is inserted using the GMC timezone.</p> <p>Valid values:</p> <ul style="list-style-type: none">• true: Treats input values as display values and they are manipulated so they are stored properly in the database.• false: Treats input values as actual values and stores them in the database without manipulation. <p>Data type: Boolean</p> <p>Default: false - This matches the data type that is returned during data retrieval (GET methods), which is the actual values.</p> <p>Note: To set the value of an encrypted field, you must set this parameter to true. If this parameter is not set to true, values submitted to encrypted fields are not saved. Additionally, the requesting user must have the appropriate encryption context prior to submitting the request. Encrypted fields are hidden for users without the appropriate encryption context. For more information on field encryption see Encryption support.</p>
sysparm_offset	<p>Starting record index for which to begin retrieving records. Use this value to paginate record retrieval. This functionality enables the retrieval of all records, regardless of the number of records, in small manageable chunks.</p> <p>For example, the first time you call this endpoint, sysparm_offset is set to "0". To simply page through all available records, use <code>sysparm_offset=sysparm_offset+sysparm_limit</code>, until you reach the end of all records.</p>

	<p>Do not pass a negative number in the sysparm_offset parameter.</p> <p>Data type: Number</p> <p>Default: 0</p>
sysparm_query	<p>Encoded query used to filter the result set. You can use a UI filter to obtain a properly encoded query.</p> <p>Syntax:</p> <p><code>sysparm_query=<col_name><operator><value>.</code></p> <ul style="list-style-type: none">• <col_name>: Name of the table column to filter against.• <operator>: Supports the following values:<ul style="list-style-type: none">• =: Exactly matches <value>.• !=: Does not match <value>.• ^: Logically AND multiple query statements.• ^OR: Logically OR multiple query statements.• LIKE: <col_name> contains the specified string. Only works for <col_name> fields whose data type is string.• STARTSWITH: <col_name> starts with the specified string. Only works for <col_name> fields whose data type is string.• ENDSWITH: <col_name> ends with the specified string. Only works for <col_name> fields whose data type is string. <p><value>: Value to match against.</p> <p>All parameters are case-sensitive. Queries can contain more than one entry, such as <code>sysparm_query=<col_name><operator><value>[<operator><col_name><operator><value>]</code>.</p>

	<p>For example:</p> <pre>(sysparm_query=caller_id=javascript:gs.getUserID()^active=true)</pre> <p>Encoded queries also support order by functionality. To sort responses based on certain fields, use the ORDERBY and ORDERBYDESC clauses in sysparm_query.</p> <p>Syntax:</p> <ul style="list-style-type: none">• ORDERBY<col_name>• ORDERBYDESC<col_name> <p>For example:</p> <pre>sysparm_query=active=true^ORDERBYnumber^ORDERBYDESCcategory</pre> <p>This query filters all active records and orders the results in ascending order by number, and then in descending order by category.</p> <p>If part of the query is invalid, such as by specifying an invalid field name, the instance ignores the invalid part. It then returns rows using only the valid portion of the query. You can control this behavior using the property glide.invalid_query.returns_no_rows. Set this property to true to return no rows on an invalid query.</p> <p>Note: The glide.invalid_query.returns_no_rows property controls the behavior of all queries across the instance, such as in lists, scripts (GlideRecord.query()), and web service APIs.</p> <p>Data type: String</p>
sysparm_view	UI view for which to render the data. Determines the fields returned in the response.

	<p>Valid values:</p> <ul style="list-style-type: none">• desktop• mobile• both <p>If you also specify the sysparm_fields parameter, it takes precedent.</p> <p>Data type: String</p>
--	--

Dot-walking in REST API requests

You can use dot-walking when specifying the sysparm_query or sysparm_fields parameters in requests to REST APIs that support those parameters.

Note: The Import Set API does not support dot-walking.

Dot-walking in sysparm_query

You can filter queries using related record values by dot-walking in the sysparm_query parameter. For example, you can retrieve all incident records where the incident **Company** has a specific **Stock symbol** value.

```
https://<instance>.service-now.com/api/now/table/incident?  
sysparm_query=company.stock_symbol=NYX
```

Dot-walking in sysparm_fields

You can view field values from multiple tables by dot-walking in the sysparm_fields parameter. For example, you can retrieve the **Name**, **Sys_id**, and **Department** of each user that has certain roles, as well as the role **Name**.

The request runs on the User Roles [sys_user_has_role] table which defines a many-to-many relationship between users and roles. The response includes field values from the User [sys_user] and Roles [sys_user_role] tables.

https://<instance>.service-now.com/api/now/table/sys_user_has_role?sysparm_fields=role%2Crole.name%2Cuser%2Cuser.name%2Cuser.sys_id%2Cuser.department&sysparm

```
{  
  "result": [  
    {  
      "user.name": "Fred Johnson",  
      "user.sys_id": "f5a3716d0f6002003a2d47bce1050ed4",  
      "role.name": "support",  
      "user.department": {  
        "display_value": "Accounting",  
        "link": "https://<instance>.service-now.com/api/now/table/cmn_department/5b3b13530f58c2003a2d47bce1050e96"  
      },  
      "role": {  
        "display_value": "support",  
        "link": "https://<instance>.service-now.com/api/now/table/sys_user_role/3d43716d0f6002003a2d47bce1050e0d"  
      },  
      "user": {  
        "display_value": "Fred Johnson",  
        "link": "https://<instance>.service-now.com/api/now/table/sys_user/f5a3716d0f6002003a2d47bce1050ed4"  
      }  
    },  
    {  
      "user.name": "Fred Johnson",  
      "user.sys_id": "f5a3716d0f6002003a2d47bce1050ed4",  
      "role.name": "asset_mgmt",  
      "user.department": {  
        "display_value": "Accounting",  
        "link": "https://<instance>.service-now.com/api/now/table/cmn_department/5b3b13530f58c2003a2d47bce1050e96"  
      },  
      "role": {  
        "display_value": "asset_mgmt",  
        "link": "https://<instance>.service-now.com/api/now/table/sys_user_role/ac73b52d0f6002003a2d47bce1050eec"  
      },  
      "user": {  
        "display_value": "Fred Johnson",  
        "link": "https://<instance>.service-now.com/api/now/table/sys_user/f5a3716d0f6002003a2d47bce1050ed4"  
      }  
    }  
  ]  
}
```

```
        }
    ]
}
```

REST API HTTP response codes

Calls made to REST endpoints return HTTP response codes. You can use these response codes to ensure that the REST API executed properly. If it did not, the endpoint returns an error response code. Use the information in the error response to troubleshoot issues with your call format. For a list of standard response codes that an endpoint may return, see [REST API HTTP response codes](#). For the list of response codes returned by a specific ServiceNow REST API, see the [REST API reference](#).

REST API security

By default, ServiceNow REST APIs use basic authentication or OAuth to authorize user access to REST APIs/endpoints. You can also configure your instance to use [multi-factor authentication](#) to access REST APIs.

The user ID that you specify in a REST endpoint call is subject to access control in the same way as an interactive user. Each request requires the proper authentication information, such as user name and password. Ensure that each endpoint request includes an Authorization header with sufficient credentials to access the endpoint.

ServiceNow REST APIs also support cookies that enable binding to the existing session.

For information on mutual authentication, see [Certificate-based authentication](#).

REST API roles

In addition to user authentication, each REST endpoint can have different requirements for the roles required to access the endpoint. Some require the admin role and others require API specific roles. Role requirements are specified in the access control list (ACL) associated with the REST API/endpoint. For specifics on the valid roles for each REST API/endpoint, refer to the [REST API reference](#) or locate the associated ACL for the API/endpoint within an instance through **System Security > Access Control (ACL)**.

REST API ACLs

REST API ACLs define criteria, such as the roles needed and conditions that a user must meet to access a ServiceNow REST API or endpoint. A single ACL may be defined for an entire REST API, such as the Table API and Attachment API ACLs, or for an individual endpoint, such as the clotho_rest_put ACL that only applies to MetricBase PUT methods.

The following ServiceNow REST API ACLs are available in the base system but are deactivated by default. All other ServiceNow REST API ACLs are active by default.

- Table API
- Aggregate API
- Import Set API
- Attachment API

For additional information on ACLs, see [Access control list rules](#).

Important: You should never modify the names of REST API ACLs.

REST API table access

By default, all tables, including base system tables, global tables, and scoped tables are accessible through web services. You must fulfill any web service security requirements, such as basic authentication and ACLs to access tables through web services. Fields for which the calling entity does not have rights to because of ACLs are not returned in a REST query response.

To allow access to tables without any authentication or authorization, add the table name to the Public Pages [sys_public] table with a status of **Active**. The REST interface still enforces any defined ACLs on associated tables. If ACL enforcement is not the desired behavior, you must deactivate the ACLs on the tables, which is not typically suggested.

You can also control direct web service access to tables using the **Allow access to this table via web services** check box on the table application access settings. You must select this check box to enable web service interaction with the table.

Note: The application access fields controlling CRUD operations, such as **Can read** or **Can create** do not apply to web service requests.

Multi-factor authentication for inbound REST

When multi-factor authentication is enabled for a user account, you must submit an MFA token with basic auth credentials when making REST requests as that user.

To send an MFA token with a REST request, append the token to the end of the user's password in the basic auth username:password string, such as `joe.employee:password62161147`. Encode the full string including the MFA token using base64 encoding, then send the encoded string in the Authorization header.

Multi-factor authentication REST responses

The response to an MFA authentication request varies depending on the validity of the provided credentials and MFA token.

Responses

Result	Description
Basic auth credentials and MFA token are valid	User is authenticated and the session created. The request is processed normally.
Basic auth credentials are valid but MFA token is invalid or missing	Response returns error 401. The response includes the X-MFA_TOKEN header with the value invalid .
Basic auth credentials are invalid	Response returns error 401. The X-MFA_TOKEN header is not included in the response.

REST API CORS support

The REST API supports cross-origin resource sharing (CORS) security. CORS support allows you to define which domains can access each REST API.

By defining a CORS rule for a domain, you can allow cross-origin requests from that domain. Cross-origin requests cannot be made from domains without a CORS rule.

Note: CORS support applies only to REST APIs, including scripted REST web services. Other web service APIs, such as the SOAP API, do not support CORS.

You can configure CORS to allow access to only certain APIs/endpoints, HTTP methods, and headers from other domains. For example, you can limit requests to the Table API from a specific domain to allow only GET operations.

To view the CORS rules defined on your instance, navigate to **System Web Services > CORS Rules**.

You can disable CORS support for an instance by setting the `glide.rest.cors.enabled` property to **false**. When **false**, no CORS evaluation is performed on incoming REST requests. This property is **true** by default.

REST API Explorer

The REST API Explorer is a ServiceNow tool that uses information from your instance to provide a list of endpoints, methods, and variables that you can use to build and send REST requests.

After you build the request, the REST API Explorer provides code samples in multiple programming languages that you can use to initiate the request, and detailed request and response information.

To access the REST API Explorer, in your instance, navigate to **System Web Services > REST API Explorer**. You must have the `rest_api_explorer` role to access the REST API Explorer. For additional information, see [Use the REST API Explorer](#).

Warning: The REST API Explorer interacts with data on the current instance. Use caution when working with requests that create, edit, or delete data on a production instance.

Automated Test Framework support

The Automated Test Framework (ATF) supports inbound REST test steps. You can create automated tests for custom inbound REST APIs that you create. Creating tests for your custom REST APIs simplifies upgrade testing, and makes it possible to verify that modifications to a REST API are backward compatible.

Example REST client applications

Several example REST client applications and source code are available to demonstrate integrations using REST web services. The example REST client applications demonstrate how to use the ServiceNow REST API with an external application, such as a NodeJS or iOS application.

Important: These applications are provided only as a demonstration of the REST functionality and are not officially supported.

The examples are open source and available to the community. You can use these example applications to help familiarize yourself with the REST functionality, or use them as a starting point to create your own REST client applications.

Users with the `rest_api_explorer` role can access the list of available client applications by navigating to **System Web Services > REST > Example Client Apps**.

When viewing the list of applications, click an application to view the source code and additional documentation hosted on GitHub.

Developer training

On the ServiceNow® Developer Site, you can get training for [Inbound REST integrations](#) and [Outbound REST integrations](#).

Additional information

The remainder of the REST API section contains "how to" topics that describe specific implementations using the ServiceNow REST API and provides reference information that describes various data elements used by the ServiceNow REST API.

- Use the REST API Explorer

In this tutorial you will use the REST API Explorer to test the ServiceNow REST APIs.

- [Inbound REST API rate limiting](#)

To prevent excessive inbound REST API requests, set rules that limit the number of inbound REST API requests processed per hour. You can create rules to limit requests for specific users, users with specific roles, or all users.

- [Debug REST queries](#)

You can debug REST queries by reviewing the session debug log.

- [Return session debug logs in a REST response](#)

You can include session debug logs in a REST response body by passing the X-WantSessionDebugMessages header in the request.

- [CORS domain requirements](#)

When you define a cross-origin resource sharing (CORS) rule, the value you enter in the **Domain** field must meet certain requirements. Each CORS rule supports a single wildcard to match incoming Origin headers.

- [Define a CORS rule](#)

You can define a CORS rule to control which domains can access specific REST API endpoints.

- [Enable OAuth with inbound REST](#)

Using OAuth, you can pass a user ID and password once, and then use a token for subsequent REST requests instead of submitting credentials with each request.

- [REST API HTTP response codes](#)

REST Messages sent to an instance return a specific HTTP response code.

- [Scripted REST APIs](#)

The scripted REST API feature allows application developers to build custom web service APIs.

Use the REST API Explorer

In this tutorial you will use the REST API Explorer to test the ServiceNow REST APIs.

The REST API Explorer allows you to discover ServiceNow REST APIs, quickly construct and execute requests, and view responses from ServiceNow REST APIs within your browser. Before beginning, ensure that your user account has the `rest_api_explorer` and `web_service_admin` roles. These roles are required to complete the example procedures.

- [Access the REST API Explorer](#)

View available REST API resources using the REST API Explorer.

- [Retrieve existing incidents](#)

Use a GET request to view existing incident records.

- [Create an incident record](#)

Use a POST request to create a new record.

- [Read the inserted incident](#)

Use the Location header from the POST response to run a GET request.

- [Update the incident](#)

Update the incident record using either a PUT or PATCH function.

- [Delete the incident](#)

Delete the incident using a DELETE request.

- [Explore the REST API for a table](#)

You can explore the REST API for a table directly from that table.

Explore the API using the REST API Explorer to quickly construct and test REST requests for that table.

- [Export to OpenAPI specification](#)

Export a REST API as an OpenAPI specification to import it into another web services tool.

View available REST API resources using the REST API Explorer.

Before you begin

Role required: web_service_admin, rest_api_explorer, or admin

About this task

You can browse available APIs, API versions, and methods for each API.

Procedure

Navigate to **All > System Web Services > REST > REST API Explorer**.

The screenshot shows the REST API Explorer interface. On the left, a sidebar lists 'Table API' and 'latest'. Below these are several blue links: 'Retrieve records from a table (GET)', 'Create a record (POST)', 'Retrieve a record (GET)', 'Modify a record (PUT)', 'Delete a record (DELETE)', and 'Update a record (PATCH)'. The main panel is titled 'Table API' and describes it as allowing CRUD operations on existing tables. It shows a 'Retrieve records from a table' section with a GET request URL: 'GET https://[REDACTED].service-now.com/api/now/table/{tableName}'. Below this is a 'Prepare request' section with 'Request Headers' for 'Request Format' (JSON), 'Response Format' (JSON), and 'Authorization' (Send as me). There are also sections for 'Path Parameters' (tableName: incident) and 'Query Parameters' (sysparm_query, sysparm_display_value, sysparm_exclude_reference_link).

Use a GET request to view existing incident records.

Before you begin

Role required: admin, web_service_admin, or rest_api_explorer

About this task

Use the REST API Explorer to send the following request:

GET https://instance.service-now.com/api/now/v1/table/incident

Procedure

1. In the top-left of the REST API Explorer, select **Table API** and version **v1**.
2. Click **Retrieve records from a table (GET)**. For information, see [Table API - GET table](#)
3. In the Path Parameters section, select the **Incident (incident)** table.
4. Scroll to the bottom of the page and click **Send**.

The response includes incident records from the instance. The REST API Explorer limits queries to 10 records at a time. Only the first 10 incident records appear. The response also includes a Link header that provides the URL to query the next 10 incident records.

The response also indicates the **Status code** and **Execution time** (in milliseconds) of the request.

Response

Status code	200 OK
Execution time (ms)	85
Headers	
Cache-control	no-cache,no-store,must-revalidate,max-age=-1
Content-encoding	gzip
Content-type	application/json;charset=UTF-8
Date	Tue, 11 Sep 2018 18:46:42 GMT
Expires	0
Link	<https://sysparm_offset=0>;rel="first",<https://sysparm_limit=1&sysparm_offset=1>;rel="next",<https://sysparm_limit=1&sysparm_offset=0>;rel="prev",<https://sysparm_limit=1&sysparm_offset=1>;rel="last"
Pragma	no-store,no-cache
Server	ServiceNow
Strict-transport-security	max-age=63072000; includeSubDomains
Transfer-encoding	chunked
X-is-logged-in	true
X-total-count	4
X-transaction-id	b955872c13e4

Response Body

```
{  
  "result": [  
    {  
      "parent": "",  
      "made_sla": "true",  
      "caused_by": "",  
      "watch_list": "",  
      "upon_reject": "cancel",  
      "sys_updated_on": "2018-08-02 16:40:36",  
      "child_incidents": "0",  
      "hold_reason": "",  
      "approval_history": "",  
      "skills": "",  
      "number": "INC0010005",  
      "resolved_by": "",  
      "sys_updated_by": "admin",
```

Use a POST request to create a new record.

Before you begin

Role required: admin, web service admin, or rest api explorer

About this task

Use the REST API Explorer to send the following request:

POST https://instance.service-now.com/api/now/v1/table/incident

Procedure

1. In the top-left of the REST API Explorer, click **Create a record (POST)**.
2. In the Path Parameters section, select the **Incident (incident)** table.
3. In the Request Body section, click **Add a field**.
4. Select a field and specify a value for that field.
5. (Optional) Click the plus sign (+) and specify any additional field to assign a value to.
The request body updates automatically based on your entries, such as { "short_description": "Test incident creation through REST", "comments": "These are my comments" }
6. After constructing the request, click **Send**.

The response includes a Location header that specifies where the incident was created and how to retrieve the incident.

Tip: Record this header to use in the next part of this guide.

The response also indicates the **Status code** and **Execution time** (in milliseconds) of the request.

Response

Status code	201 Created
Execution time (ms)	1258
Headers	
Cache-control	no-cache,no-store,must-revalidate,max-age=-1
Content-encoding	gzip
Content-type	application/json;charset=UTF-8
Date	Tue, 11 Sep 2018 19:00:45 GMT
Expires	0
Location	https://[REDACTED]/api/now/v1/table/incident/ /af888bac13e42300fc9c32228144b0d4
Pragma	no-store,no-cache
Server	ServiceNow
Strict-transport-security	max-age=63072000; includeSubDomains
Transfer-encoding	chunked
X-is-logged-in	true
X-transaction-id	938887ac13e4

Response Body

```
{  
  "result": {  
    "parent": "",  
    "made_sla": "true",  
    "caused_by": "",  
    "watch_list": "",  
    "upon_reject": "cancel",  
    "sys_updated_on": "2018-09-11 19:00:45",  
    "child_incidents": "0",  
    "hold_reason": "",  
    "approval_history": "",  
    "skills": "",  
    "number": "INC0010009",  
    "resolved_by": "",  
    "sys_updated_by": "terry.barracough@snc",  
    "opened_by": {  
      "name": "Terry Barracough",  
      "id": "terry.barracough@snc",  
      "type": "User"  
    }  
  }  
}
```

Use the Location header from the POST response to run a GET request.

Before you begin

Role required: admin, web_service_admin, or rest_api_explorer

About this task

Use the REST API Explorer to send the following request:

GET https://instance.service-now.com/api/now/v1/table/incident/
{sys_id}

Procedure

1. In the top-left of the REST API Explorer, click **Retrieve a record (GET)**.
2. In the Path Parameters section, select the incident table.
3. In the **sys_id** field, enter the sys_id of the record you created.

The record sys_id appears as a 32-character string at the end of the POST response Location header.

4. Click **Send**.

The response body contains a text representation of the record. You can control the format of the response, such as JSON or XML, using the **Response Format** field.

The response also indicates the **Status code** and **Execution time** (in milliseconds) of the request.

Response

Status code	200 OK
Execution time (ms)	90
Headers	
Cache-control	no-cache,no-store,must-revalidate,max-age=-1
Content-encoding	gzip
Content-type	application/json;charset=UTF-8
Date	Tue, 11 Sep 2018 20:48:40 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Strict-transport-security	max-age=63072000; includeSubDomains
Transfer-encoding	chunked
X-is-logged-in	true
X-transaction-id	1c41afa81328

Response Body

```
{  
  "result": {  
    "parent": "",  
    "made_sla": "true",  
    "caused_by": "",  
    "watch_list": "",  
    "upon_reject": "cancel",  
    "sys_updated_on": "2018-09-11 19:00:45",  
    "child_incidents": "0",  
    "hold_reason": "",  
    "approval_history": "",  
    "skills": "",  
    "number": "INC0010009",  
    "resolved_by": "",  
    "sys_updated_by": "terry.barracough@snc",  
    "opened_by": {  
      "name": "Terry Barracough",  
      "id": "1c41afa81328",  
      "type": "User"  
    }  
  }  
}
```

Update the incident record using either a PUT or PATCH function.

Before you begin

Role required: admin, web_service_admin, or rest_api_explorer

About this task

Use the REST API Explorer to send the following request:

```
PUT https://instance.service-now.com/api/now/v1/table/incident/  
{sys_id}?sysparm_exclude_ref_link=true
```

Procedure

1. In the top-left of the REST API Explorer, click **Modify a record (PUT)** or **Update a record (PATCH)**.
2. In the Path Parameters section, select the **Incident (incident)** table.
3. In the **sys_id** field, enter the sys_id of the record you created.
4. In the Request Body section, click **Add a field**.
5. Select the **Short description** field and specify a new value.
6. Click **Send**.

The response indicates the **Status code** and **Execution time** (in milliseconds) of the request.

Response

Status code	200 OK
Execution time (ms)	284
Headers	
Cache-control	no-cache,no-store,must-revalidate,max-age=-1
Content-encoding	gzip
Content-type	application/json;charset=UTF-8
Date	Tue, 11 Sep 2018 20:59:08 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Strict-transport-security	max-age=63072000; includeSubDomains
Transfer-encoding	chunked
X-is-logged-in	true
X-transaction-id	31a3232c1328

Response Body

```
"reopened_time": "",  
"resolved_at": "",  
"approval_set": "",  
"subcategory": "",  
"work_notes": "",  
"short_description": "This is a different short description",  
"close_code": "",  
"correlation_display": "",  
"delivery_task": "",  
"work_start": "",  
"assignment_group": "",  
"additional_assignee_list": "",  
"business_stc": "",  
"description": "",  
"calendar_duration": "",  
"close_notes": "",  
"next_fix": "1"
```

- Verify that the Response Body contains the updated **short_description** value.

Delete the incident using a DELETE request.

Before you begin

Role required: admin, web_service_admin, or rest_api_explorer

About this task

Use the REST API Explorer to send the following request:

```
DELETE https://instance.service-now.com/api/now/v1/table/incident/{sys_id}
```

Procedure

1. In the top-left of the REST API Explorer, click **Delete a record (DELETE)**.
2. In the Path Parameters section, select the **Incident (incident)** table.
3. In the **sys_id** field, enter the sys_id of the record you created.
4. Click **Send**.

The response indicates the **Status code** and **Execution time** (in milliseconds) of the request.

Response

Status code	204 No Content
-------------	----------------

Execution time (ms) 2689

Headers

Content-encoding	gzip
Date	Tue, 11 Sep 2018 21:10:59 GMT
Server	ServiceNow
Strict-transport-security	max-age=63072000; includeSubDomains
X-is-logged-in	true
X-transaction-id	2a56ab6c1328

Response Body

...

5. Verify that the response **Status code** is 204.

You can explore the REST API for a table directly from that table. Explore the API using the REST API Explorer to quickly construct and test REST requests for that table.

Before you begin

Role required: itil, personalize_dictionary, and rest_api_explorer

Procedure

1. Navigate to any form or list.
2. In a form, right-click the form header and select **Configure > Table**.
3. In a list, perform the appropriate action for the list version.
 - For List v2, right-click any column heading and select **Configure > Table**.
 - For List v3, open the list title menu and select **Configure**, and then click **Table**.
4. Click the **Explore REST API** related link.
The REST API Explorer opens, displaying the Table API for the selected table. If the table does not allow web service interaction, a warning appears instead.

What to do next

Use the REST API Explorer to construct and test REST requests for the table.

Export a REST API as an OpenAPI specification to import it into another web services tool.

Before you begin

Role required: web_service_admin, rest_api_explorer, or admin

Procedure

1. Navigate to **All > System Web Services > REST > REST API Explorer**.
2. Select the API you want to export.
For example, use these choices to select the Table API for export.

Field	Value
Namespace	now
API Name	Table API

Field	Value
API Version	latest

3. Below the list of API methods, select the link for the export format you want to use.

Option	Description
Export OpenAPI Specification (YAML)	Format the export in YAML.
Export OpenAPI Specification (JSON)	Format the export in JSON.

The browser displays a window to select a download location.

4. Select the download location and file name.
5. Save the file.

Result

The system downloads the API in the format you selected.

What to do next

Import the API into a web services tool such as Postman or Insomnia.

Inbound REST API rate limiting

To prevent excessive inbound REST API requests, set rules that limit the number of inbound REST API requests processed per hour. You can create rules to limit requests for specific users, users with specific roles, or all users.

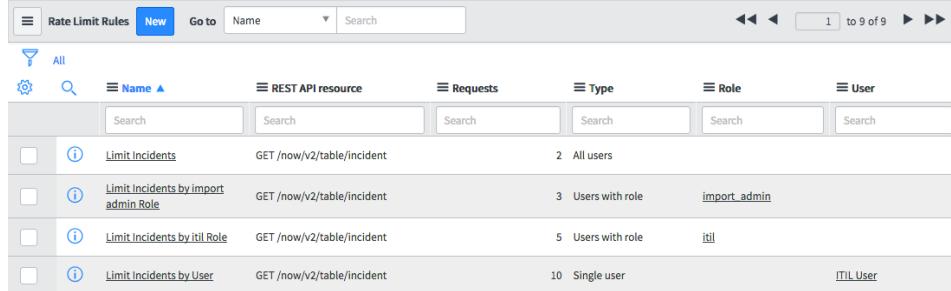
Note: As requests reach an instance, each node maintains a rate limit count per user. Every 30 seconds, the count is committed to the database. As a result, a rate limit rule may not take effect for up to 30 seconds.

Rate limiting priority

If an inbound REST API request matches multiple rate limit rules for the same resource, rate limiting priority is enforced as follows:

- Rules set for **Single user** override rules for **All users** and rules for **Users with role**.
- Rules set for **Users with role** override rules for **All users**.

In this example, there are four rate limit rules for the same REST API resource: GET /now/v2/table/incident:



	Name	REST API resource	Requests	Type	Role	User
<input type="checkbox"/>	Limit Incidents	GET /now/v2/table/incident	2	All users		
<input type="checkbox"/>	Limit Incidents by import admin Role	GET /now/v2/table/incident	3	Users with role	import_admin	
<input type="checkbox"/>	Limit Incidents by itil Role	GET /now/v2/table/incident	5	Users with role	itil	
<input type="checkbox"/>	Limit Incidents by User	GET /now/v2/table/incident	10	Single user		ITIL User

These rate limit rules are applied in the following order:

1. **Limit Incidents by User** applies to ITIL User, who can submit up to 10 requests per hour.
2. **Limit Incidents by import admin Role** applies to each user with the import_admin role. Each user with the import_admin role can submit up to three requests per hour.
3. **Limit Incidents by itil Role** applies to each user with the itil role. Each user with the itil role can submit up to five requests per hour.
4. **Limit Incidents** applies to all users. Each user can submit up to two requests per hour.

When ITIL User makes the request GET /now/v2/table/incident, the request matches the criteria for three rules: **Limit Incidents**, **Limit Incidents by itil Role**, and **Limit Incidents by User**. Only the **Limit Incidents by User** rule is applied because it takes precedence over the other rules. As a result, ITIL User can submit a maximum of 10 requests per hour.

If a user has two or more roles matching the criteria of multiple rate limiting rules for a REST API resource, the rule allowing the lowest number

of requests applies to the user's requests for the resource. For the example rules in the figure above, assume that user Abel Tuter has both the import_admin role and the itil role. When Abel Tuter submits a request, it meets the criteria for both the **Limit Incidents by admin Role** rule and the **Limit Incidents by itil Role** rule. Only the **Limit Incidents by admin Role** rule is applied because it allows the lowest number of requests. As a result, Abel Tuter can submit a maximum of three requests per hour.

REST API response headers

You can generate inbound REST API requests using the [Use the REST API Explorer](#) or an HTTP client, such as Postman. If the request matches a rate limit rule, several HTTP response headers provide information about rate limiting:

- **X-RateLimit-Limit** displays the number of requests allowed per hour.
- **X-RateLimit-Reset** displays the Unix time until the next scheduled reset.
- **X-RateLimit-Rule** displays the sys_id of the rate limit rule that is being enforced.

Body	Cookies (4)	Headers (19)	Test Results	Status: 200 OK	Time: 1151 ms	Size: 167.22 KB
<pre>Cache-control → no-cache,no-store,must-revalidate,max-age=-1 Content-Encoding → gzip Content-Type → application/json;charset=UTF-8 Date → Wed, 28 Mar 2018 21:11:01 GMT Expires → 0 Pragma → no-store,no-cache Server → ServiceNow Set-Cookie → glide_user=Secure; Max-Age=0; Expires=Thu, 01-Jan-1970 00:00:10 GMT; Path=/; HttpOnly Set-Cookie → glide_user_session=Secure; Max-Age=0; Expires=Thu, 01-Jan-1970 00:00:10 GMT; Path=/; HttpOnly Set-Cookie → glide_user_route=glide.f0ed3265b4b8434b7cd874c7385b11e0;Secure; Max-Age=2147483647; Expires=Tue, 16-Apr-2086 00:25:08 GMT; Path=/; HttpOnly Set-Cookie → glide_session_store=F474E77E13091300592F54022244B0AF;Secure; Max-Age=1800; Expires=Wed, 28-Mar-2018 21:41:01 GMT; Path=/; HttpOnly Strict-Transport-Security → max-age=63072000; includeSubDomains Transfer-Encoding → chunked X-Is-Logged-In → true X-RateLimit-Limit → 10 X-RateLimit-Reset → 1522274400 X-RateLimit-Rule → baa1d22613091300592f54022244b0c5</pre> <p>X-Total-Count → 55 X-Transaction-ID → b3d4eb7e1309</p>						

If a request is denied because it exceeds the rate limit, the system returns a **Retry After** response header in addition to the response headers about rate limiting. The **Retry After** response header displays the number of seconds after which you can retry the request to avoid exceeding the rate limit. The following error response is returned:

```
{
  "error": {
    "message": "Rate limit exceeded",
    "detail": "Rate limit of 10 requests per hour for Table API exceeded"
  },
  "status": "failure"
}
```

The status of a denied request is 429 Too Many Requests.

Body	Cookies (5)	Headers (16)	Test Results
<pre>Cache-control → no-cache,no-store,must-revalidate,max-age=-1 Content-Encoding → gzip Content-Type → application/json;charset=UTF-8 Date → Wed, 28 Mar 2018 22:00:14 GMT Expires → 0 Pragma → no-store,no-cache Retry-After → 3585</pre>			Status: 429 Too Many Requests Time: 90 ms Size: 805 B

- [Create an inbound REST API rate limit](#)

Create rate limit rules to limit the number of inbound REST API requests processed per hour.

- [Reset an inbound REST API rate limit](#)

Reset a rate limit rule to reset the rate limit count to zero (0) and delete any violations for the current hour.

- [Monitor inbound REST API rate limit counts and violations](#)

To determine if you have set a rate limit rule appropriately, monitor the counts and violations for inbound REST API requests that are restricted by the rule.

- [Investigate inbound REST API rate limit violations](#)

Investigate rate limit violations to determine which rate limit rules are being exceeded and which users are exceeding those rate limits.

Create rate limit rules to limit the number of inbound REST API requests processed per hour.

Before you begin

Role required: rate_limit_admin

About this task

Set rate limits for all users, users with specific roles, or all users.

Note: As requests reach an instance, each node maintains a rate limit count per user. Every 30 seconds, the count is committed to the database. As a result, a rate limit rule may not take effect for up to 30 seconds.

Procedure

1. Navigate to **All > System Web Services > REST > Rate Limit Rules**.
2. Click **New** and enter the following field values.

REST API Rate Limit Rule form

Field	Description
REST API resource	Value derived from the values entered at the following fields.
Name	Unique name for the rate limit rule.
REST API	REST API selected from the list of all external-facing REST APIs for the instance.
Version	Version of the REST API . Values listed depend on the REST API selected.
Resource	Resource for the Version . Values listed depend on the Version selected.
Table	Table that you want to target. Appears only when you select Table API as the REST API .
Import set table	Import set table that you want to target. Appears only when you select Import Set API as the REST API .
Active	Check box to indicate that the rate limit rule is active.

Field	Description
	Rate limit rules are activated by default as soon as you create them. You can deactivate rate limit rules to stop enforcing a rate limit or activate rate limit rules to resume enforcing a rate limit.
Request limit per hour	<p>Maximum number of requests allowed per hour.</p> <p>Note: Whenever you update the value of this field, the Now Platform resets the count of requests to 0 and deletes all violations for the current hour.</p>
Apply to	<p>Users restricted by this rule:</p> <ul style="list-style-type: none"> • Single user applies the rate limit to a specific user. • Users with role applies the rate limit to all users with a specific role. • All users applies the rate limit to all users.
Role	Role to which the rate limit applies. Appears only when you select Users with role at the Apply to field.
User	User to whom the rate limit applies. Appears only when you select Single user at the Apply to field.

3. Click **Submit**.
The new rate limit goes into effect.

What to do next

After you submit the rule, the Now Platform adds the following related lists to the rule record:

Rate Limit Counts

Lists, by user, the number of inbound REST API requests affected by this rate limit rule.

Rate Limit Violations

Lists, by user, the violations of this rate limit rule.

You can use these related lists to [Monitor inbound REST API rate limit counts and violations](#).

Reset a rate limit rule to reset the rate limit count to zero (0) and delete any violations for the current hour.

Before you begin

Role required: rate_limit_admin

Procedure

1. Navigate to **All > System Web Services > REST > Rate Limit Rules**.
2. Select the rate limit rule for which you want to reset the rate limit count.
3. Click the **Reset Rate Limit Counts** related link.

Result

For the current hour, the system resets the rate limit count for the rate limit rule to zero (0) and removes all violations. The system begins incrementing the rate limit counts and violations as REST API requests are received for processing.

To determine if you have set a rate limit rule appropriately, monitor the counts and violations for inbound REST API requests that are restricted by the rule.

Before you begin

Role required: rate_limit_admin

Procedure

1. Navigate to **All > System Web Services > REST > Rate Limits**.
2. Select the rate limit rule for which you want to monitor rate limit counts and violations.
3. View the information in the following related lists:
 - In the Rate Limit Counts related list, view the count of inbound REST API requests limited by the rule. This list is cleared daily.
 - In the Rate Limit Violations related list, view the number of inbound REST API requests that exceeded the **Request limit per hour** value for the rule. This list is cleared biweekly.

Investigate rate limit violations to determine which rate limit rules are being exceeded and which users are exceeding those rate limits.

Before you begin

Role required: rate_limit_admin

About this task

Violations are created when inbound REST API requests reach the maximum allowed request count for a specific REST API for that hour.

Procedure

1. Navigate to **All > System Web Services > REST > Rate Limit Violations**.
2. On the Rate Limit Violations page, select the rate limit rule that you want to investigate.

3. On the Rate Limit Violations related list, review the violations by user.
This list is cleared biweekly.

What to do next

You may need to reevaluate the **Request limit per hour** value for a rate limit rule, depending on the number of violations of that rule. You may also need to educate users about rate limits, depending on how many times specific users violate rate limit rules.

Debug REST queries

You can debug REST queries by reviewing the session debug log.

When the glide.rest.debug property is **true**, all REST processing is logged in the session debug log.

REST Logging includes processing durations, headers, and the request body. Prolonged use of this property can affect performance, so it is best to use it while debugging REST processing, and then set the property back to **false**.

Note: You may not see the resulting log statements if you are not on the application node that processed your REST request. In this case, please contact Technical Support.

You can include session debug logs in a REST response body by passing the X-WantSessionDebugMessages header in the request. For more information, see [Returning session debug logs in a REST response](#). To view debug logs, see [Display debugging logs](#).

Sample log output

```
2014-03-19 11:10:37 (633) http-12 New transaction 083A603  
1D7231100261253B2B252035C #28 /api/now/table/incident  
2014-03-19 11:10:37 (653) REST API-thread-1 SYSTEM DEBUG  
: [REST API] RESTAPIProcessor : Started initializing REST  
Request  
2014-03-19 11:10:37 (653) REST API-thread-1 SYSTEM DEBUG  
: [REST API] RESTAPIProcessor : Request Method:POST  
2014-03-19 11:10:37 (656) REST API-thread-1 SYSTEM DEBUG  
: [REST API] RESTAPIProcessor : Request Header: host:local  
host:8080
```

```
2014-03-19 11:10:37 (656) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: user-agent
:Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:12.0) Gec
ko/20100101 Firefox/12.0
2014-03-19 11:10:37 (656) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: accept:app
lication/json
2014-03-19 11:10:37 (656) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: accept-enc
oding:gzip, deflate
2014-03-19 11:10:37 (656) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: connection
:keep-alive
2014-03-19 11:10:37 (657) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: content-typ
e:application/json; charset=UTF-8
2014-03-19 11:10:37 (657) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: content-le
ngth:31
2014-03-19 11:10:37 (657) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: cookie:gli
de_user_route=glide.20e7f4cd6bdc0d444810117aacc0eeae; JSES
SIONID=F07CE6ACF8AF237CB239AF43B7F360BF; glide_user="U0N2M
jo0MDNhNjAzMWQ3MjMxMTAwMjYxMjUzYjJiMjUyMDM2OTo2ODE2Zjc5Y2M
wYTgwMTY0MDFjNWEzM2J1MDRiZTQ0MQ=="; glide_user_session="U0
N2Mjo0MDNhNjAzMWQ3MjMxMTAwMjYxMjUzYjJiMjUyMDM2OTo2ODE2Zjc5
Y2MwYTgwMTY0MDFjNWEzM2J1MDRiZTQ0MQ=="
2014-03-19 11:10:37 (657) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: pragma:no
-cache
2014-03-19 11:10:37 (657) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Request Header: cache-cont
rol:no-cache
2014-03-19 11:10:38 (357) REST API-thread-1 SYSTEM [REST
API] RouteRegistry : Loaded Routes to Cache
2014-03-19 11:10:38 (357) REST API-thread-1 SYSTEM DEBUG
: [REST API] RouteRegistry : Route loading time 0:00:00.10
5
2014-03-19 11:10:38 (357) REST API-thread-1 SYSTEM DEBUG
: [REST API] URIHandler : Resolving URI: /now/table/incide
nt
2014-03-19 11:10:38 (391) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : URI Resolving Duration 387
570:10:38.391
```

```
2014-03-19 11:10:38 (424) REST API-thread-1 SYSTEM DEBUG
: [REST API] RESTAPIProcessor : Finished initializing RES
T Request
2014-03-19 11:10:38 (540) REST API-thread-1 083A6031D723
1100261253B2B252035C #28 /api/now/table/incident Parameter
s -----
    api=api
2014-03-19 11:10:38 (541) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] RESTAPIProcessor : P
rocessing REST Request /api/now/table/incident
2014-03-19 11:10:38 (541) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] RESTAPIProcessor : P
re-Service processing duration 0:00:00.000
2014-03-19 11:10:38 (548) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] ServiceHandler : Inv
oking Service TableAPIService
2014-03-19 11:10:38 (552) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] TableAPIService : In
serting record
2014-03-19 11:10:38 (560) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] RequestDeserializer
: Incoming Request Body:{"short_description":"test me"}
2014-03-19 11:10:39 (508) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] TableAPIService : Gl
ide Record Insert Duration 0:00:00.956
2014-03-19 11:10:39 (508) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] TableAPIService : Qu
erying for inserted record
2014-03-19 11:10:39 (513) REST API-thread-1 083A6031D7231
100261253B2B252035C ##### Compiler Stats #####
2014-03-19 11:10:39 (514) REST API-thread-1 083A6031D7231
100261253B2B252035C Compiles: 600, time: 1,130ms
2014-03-19 11:10:39 (514) REST API-thread-1 083A6031D7231
100261253B2B252035C Cache name: "syscache_expression", max
: 8,192, size: 599, seeks: 34,182, hits: 32,980, misses: 1
,202, flushed: 0
2014-03-19 11:10:39 (514) REST API-thread-1 083A6031D723
1100261253B2B252035C Total classes: 1,402, bytecode length
: 3,263,462
2014-03-19 11:10:39 (514) REST API-thread-1 083A6031D7231
100261253B2B252035C Total loaders created: 601, unloaded:
0, existing: 601
2014-03-19 11:10:39 (652) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] TableAPIService : Cr
```

```
eating service result for insert request
2014-03-19 11:10:39 (655) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] ServiceHandler : End of Service InvocationTableAPIService
2014-03-19 11:10:39 (655) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] ServiceHandler : Service Invocation Duration 0:00:01.107
2014-03-19 11:10:39 (660) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] ServiceHandler : Serializing Response
2014-03-19 11:10:39 (706) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] ServiceHandler : End of Response Serialization
2014-03-19 11:10:39 (706) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] ServiceHandler : Response Serialization Duration 0:00:00.046
2014-03-19 11:10:39 (706) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] RESTAPIProcessor : Service handling duration 0:00:01.165
2014-03-19 11:10:39 (706) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] RESTAPIProcessor : End of Request Processing
2014-03-19 11:10:39 (706) REST API-thread-1 083A6031D7231
100261253B2B252035C DEBUG: [REST API] RESTAPIProcessor : REST Request Processing time 0:00:01.165
2014-03-19 11:10:39 (707) REST API-thread-1 083A6031D7231
100261253B2B252035C #28 /api/now/table/incident -- total transaction time: 0:00:02.074, total wait time: 0:00:00.001,
, session wait: 0:00:00.000, semaphore wait: 0:00:00.001,
source: 127.0.0.1
```

Return session debug logs in a REST response

You can include session debug logs in a REST response body by passing the X-WantSessionDebugMessages header in the request.

To return session debug messages when session debugging is enabled for the current session, set the header X-WantSessionDebugMessages to **true** in the REST request.

Note: You must enable session debugging before sending this header.

Example: Request

This example demonstrates a Table API request made using REST API Explorer with Session Debug SQL enabled.

```
GET api/now/table/incident/9c573169c611228700193229fff7240
0
X-WantSessionDebugMessages:true
Content-Type: application/json; charset=UTF-8
Accept: application/json, text/plain,/*
X-UserToken: <user token>
```

Example: Response body

```
{
  "result": {
    "description": "User can't access email on mail.company.com.",
    "number": "INC0000001"
  },
  "session": {
    "debug_logs": [
      {
        "type": "sql",
        "customerUpdate": false,
        "line": "17:17:27.777: Time: 0:00:00.000 for: glide_master_db[glide.5] ..... sys_user_session0.`id` = '3BEA7001EB230200C46AC2EEF106FE2A'</span>",
        "debugClassSet": ""
      },
      {
        "type": "sql",
        "customerUpdate": false,
        "line": "17:17:27.779: Time: 0:00:00.002 for: glide_master_db[glide.6] ... `sys_id` = '7fea7001eb230200c46ac2eeef106fe2a'</span>",
        "debugClassSet": ""
      }
    ]
  }
}
```

Related concepts

- [Session debug](#)

CORS domain requirements

When you define a cross-origin resource sharing (CORS) rule, the value you enter in the **Domain** field must meet certain requirements. Each CORS rule supports a single wildcard to match incoming Origin headers.

Requirements

The value you enter in the **Domain** field must meet the following requirements.

- Begins with HTTP:// or HTTPS://.
- Is a domain pattern or IP address.
- Ends with alphanumeric characters preceded by a period, such as .com.
- Includes at most a single wildcard character immediately following the scheme and hierarchical portion of the domain pattern.

Wildcard

You can use a single wildcard character (*) in the domain pattern. Use this wildcard immediately following the scheme and hierarchical portion of the domain pattern, such as http://*.domain.com to include all subdomains. The wildcard must immediately follow the scheme and hierarchical portion of the domain pattern. If you use an IP address instead of domain pattern, you must enter the full IP address without a wildcard.

Note: You cannot use multiple wildcards, or specify a wildcard without a domain pattern. Values such as * or *.* are not supported.

Domain matching

When evaluating the Origin header in a request, ServiceNow prioritizes rules that match the domain pattern exactly. If no exact match is found, the next closest match is used.

For example, if there are rules for the domain patterns `http://*.blog.mysite.com` and `http://*.mysite.com`, a request from `http://alice.blog.mysite.com` will match the `http://*.blog.mysite.com` pattern.

Examples of valid and invalid domains

Examples

Valid domain	Invalid domain
<code>http://*.ms.net</code>	<code>https://*com</code>
<code>https://*.ms.com</code>	<code>http://*..com</code>
<code>https://*.com.au</code>	<code>http://192.168.1.*</code>
<code>http://192.168.1.1</code>	<code>http://*.168.1.126</code>
<code>http://*.service-now.com</code>	<code>http://blog.*.service-now.com</code>
<code>http://*.com</code>	<code>http://*com</code>

Define a CORS rule

You can define a CORS rule to control which domains can access specific REST API endpoints.

Before you begin

Role required: `web_service_admin`

Procedure

1. Navigate to **All > System Web Services > REST > CORS Rules**.
2. Click **New**.
3. Populate the form.

Fields

Field	Description
REST API	Select the REST API this CORS rule applies to, such as the Table API.
Domain	Enter the domain that this CORS rule applies to. This CORS rule is evaluated against requests from the specified domain. You can specify a domain pattern or an IP address. When using a domain pattern you can specify a single wildcard to match incoming origin headers.
HTTP Methods	Select the HTTP methods allowed. Only the selected methods can be called from the specified domain.
HTTP Headers	Enter a comma-separated list of HTTP headers to send in the response. Specified headers are added to the Access-Control-Expose-Headers header.
Max age	Enter the number of seconds to cache the client session. After an initial CORS request, further requests from the same client within the specified time do not require a preflight message. If you do not specify a value, the default value of 0 indicates that all requests require a preflight message.

-
4. Click **Submit**.

Enable OAuth with inbound REST

Using OAuth, you can pass a user ID and password once, and then use a token for subsequent REST requests instead of submitting credentials with each request.

About this task

OAuth can improve system security by reducing the number of times you submit user credentials. You can use OAuth to authenticate REST requests.

This video demonstrates how to authenticate to REST APIs using OAuth.

Procedure

1. Activate the OAuth 2.0 plugin.
2. Set the system property com.snc.platform.security.oauth.is.active to true.
3. Navigate to **System OAuth > Application Registry**.
4. Click **New** and then click **Create an OAuth API endpoint for external clients**.
5. Record the **client_id** and **client_secret** values from the previous step to use when requesting an access token.
6. To get an access token, use your REST client, such as CURL or Postman, to send a request to the OAuth endpoint (`oauth_token.do`).

Format the request as a URL-encoded HTTP POST body and include the required parameters.

7. Record the access token and refresh token from the response.
8. Submit the access token with subsequent REST requests.

- [REST OAuth example](#)

This example shows how to authenticate an inbound REST request using OAuth.

This example shows how to authenticate an inbound REST request using OAuth.

In this example, the OAuth token has a `client_id` of `a329c4515612210071a5e0c298ee2be8` and a `client_secret` of `password22`.

Getting an access token

```
curl -d "grant_type=password&client_id=a329c4515612210071a5e0c298ee2be8&client_secret=password22&username=RESTUser&password=RESTUserPassword" https://<instance>.service-now.com/oauth_token.do
```

Sample token response

```
{  
  "scope": "useraccount",  
  "token_type": "Bearer",  
  "expires_in": 1799,  
  "refresh_token": "jZPdkEVrWvtMjrspldNjIS0uWM4D7QV9mgmcQXD  
Vo5Qa_GVvmdR6NOp7OM038EHJnd6nZpWocFer1NcJz4zwdw",  
  "access_token": "2wRlsRCT2SYjCCJP91kwo2EFzj5qg4O3I3aC09e0  
-0hz6Ib3YK7If-LMiNorNuglfqbkL4AfkyC92KYHUCcbpQ"  
}
```

REST request with OAuth token

```
curl -H "Accept:application/json" -H "Authorization:Beare  
r 2wRlsRCT2SYjCCJP91kwo2EFzj5qg4O3I3aC09e0-0hz6Ib3YK7If-LM  
iNorNuglfqbkL4AfkyC92KYHUCcbpQ" "https://<instance>.servi  
ce-now.com/api/now/table/incident"
```

REST API HTTP response codes

REST Messages sent to an instance return a specific HTTP response code.

Status Code	Message	Details
200	Success	Success with response body.
201	Created	Success with response body.
204	Success	Success with no response body.
400	Bad Request	The request URI does not match the APIs in the system, or the operation failed for unknown reasons. Invalid headers can also cause this error.
401	Unauthorized	The user is not authorized to use the API.
403	Forbidden	The requested operation is not permitted for the user. This error can also be caused by ACL failures, or business rule or data policy constraints.
404	Not found	The requested resource was not found. This can be caused by an ACL constraint or if the resource does not exist.
405	Method not allowed	The HTTP action is not allowed for the

Status Code	Message	Details
		requested REST API, or it is not supported by any API.
406	Not acceptable	The endpoint does not support the response format specified in the request Accept header.
415	Unsupported media type	The endpoint does not support the format of the request body.

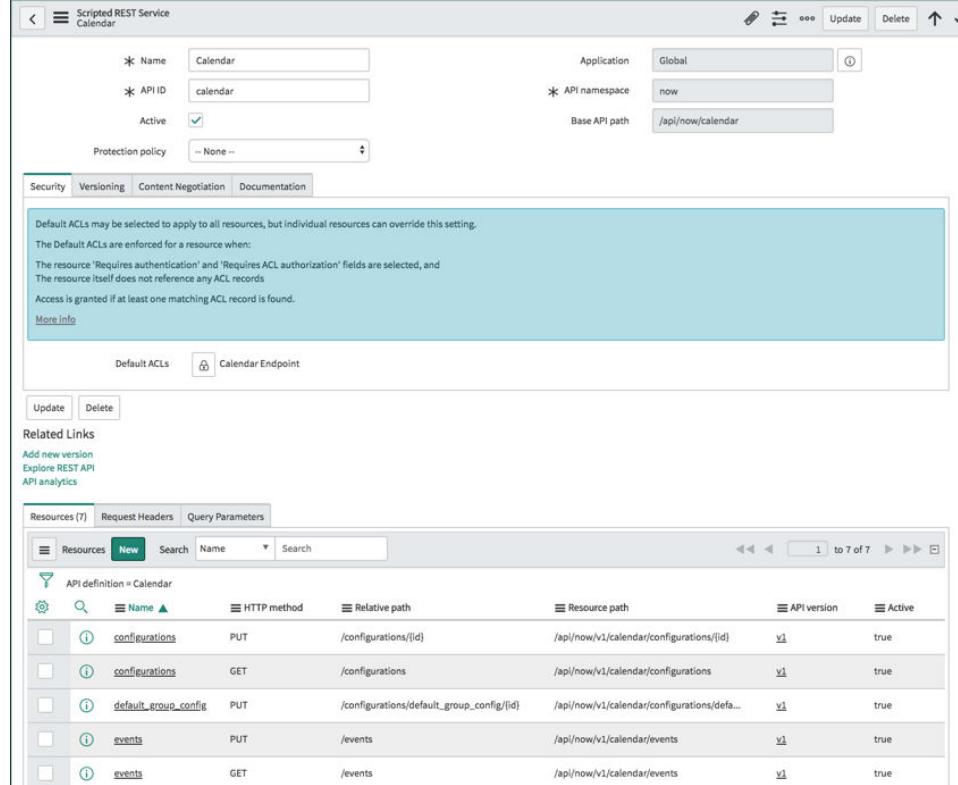
Scripted REST APIs

The scripted REST API feature allows application developers to build custom web service APIs.

You can define service endpoints, query parameters, and headers for a scripted REST API, as well as scripts to manage the request and response.

Scripted REST APIs generally follow the REST architecture, but you can customize them to use different conventions. You define scripted REST APIs using the Scripted REST Service form found under **Scripted Web Services** → **Scripted REST APIs**.

Scripted REST Service form



The screenshot shows the 'Scripted REST Service' configuration page for a service named 'Calendar'. Key settings include:

- Name:** Calendar
- Application:** Global
- API ID:** calendar
- API namespace:** now
- Base API path:** /api/now/calendar
- Active:** checked
- Protection policy:** None

The 'Default ACLs' section notes that Default ACLs are enforced for a resource when 'Requires authentication' and 'Requires ACL authorization' are selected. It also states that access is granted if at least one matching ACL record is found.

The 'Resources' section lists the following endpoints:

Name	HTTP method	Relative path	Resource path	API version	Active
configurations	PUT	/configurations/{id}	/api/now/v1/calendar/configurations/{id}	v1	true
configurations	GET	/configurations	/api/now/v1/calendar/configurations	v1	true
default_group_config	PUT	/configurations/default_group_config/{id}	/api/now/v1/calendar/configurations/defa...	v1	true
events	PUT	/events	/api/now/v1/calendar/events	v1	true
events	GET	/events	/api/now/v1/calendar/events	v1	true

The following podcast offers additional information on the use of scripted REST APIs.

Scripted REST API URIs

Scripted REST API URIs have the following format:

`https://<instance.service-now.com>/api/<name_space>/<version>/<api_id>/<relative_path>`

In this URI:

- <instance.service-now.com>: Path to the ServiceNow instance where users access the scripted REST API.
- <name_space>: For web services in the global scope, the name space is the value of the property glide.appcreator.company.code. For web

services in a scoped application, the name space is the scope name, such as x_company_appname. For additional information on name spaces, see [Application scope](#).

- <version>: Optional. Version of the endpoint to access if the API uses versioning, such as **v1**. You can access the default version of a versioned API by specifying the URL without a version number.
- <api_id>: Value of the **API ID** field on the Scripted REST Service form. By default this value is based on the service name.
- <relative_path>: Relative path defined for the resource in the Scripted REST Service form. Specifying a relative resource path allows you to have multiple resources using the same HTTP method, such as GET, in one web service. For example, a resource may specify the path /{id} when the web service has only one GET resource, or /user/{id} and /message/{id} when the web service has different resources for requesting user and message records.

Scripted REST API versioning

Scripted REST API URLs may include a version number, such as /api/management/v1/table/{tableName}. Version numbers identify the endpoint version that a URI accesses. By specifying a version number in your URLs, you can test and deploy changes without impacting existing integrations.

Default API version

A version may be marked as default. Specifying a default version allows users to access that version using a scripted REST endpoint without a version number. If no version is marked as default, the latest version is used as the default.

Scripted REST API resources

A scripted REST API resource is equivalent to a REST endpoint. It defines the HTTP method to execute, the processing script, and any override settings from the parent API. You can define one or more resources per API.

Scripted REST API query parameters

Query parameters define values that requesting users can pass in a request. When creating a scripted REST API, you can specify which parameters are available and which are mandatory for each request. You can also associate a query parameter with multiple resources.

Access request parameters in scripts using the `request` object `queryParams` field.

Scripted REST API roles

To work with scripted REST APIs, you must have the web service administrator [web_service_admin] role. Users with this role can read, create, modify, and delete scripted REST APIs and web service resources.

Note: These roles are not required to access a scripted REST API endpoint.

Request and response formats

By default, all resources in an API support the following request and response formats: `application/json`, `application/xml`, and `text/xml`. You can override the default formats at the API level. The new formats apply to all resources belonging to the API, unless an individual resource overrides the defaults.

Scripted REST API security

You can configure your scripted REST APIs with the necessary level of security. From public APIs/endpoints that don't require any security to highly secure APIs/endpoints that require user authentication with tight access control to all resources.

Scripted REST API access controls

Access control lists (ACLs) define criteria, such as the roles needed and conditions that a user must meet to access a scripted REST API or endpoint. A requesting user must satisfy at least one of the ACLs. It is not necessary to satisfy all selected ACLs. You can define a single ACL for an entire REST API or for an individual endpoint.

Note: By default, scripted REST APIs contain an ACL that prohibits users with the snc_external role from making requests to the API.

When defining a scripted REST API ACL, it must have the **Type** value **REST_Endpoint**.

For additional information on ACLs, see [Access control list rules](#) and [Configure a scripted REST API resource to require an ACL](#).

Scripted REST API security matrix

There are multiple possible security configurations for scripted REST APIs. Use this table to identify the scripted REST API security configuration that best suits your needs, and the field values to implement that configuration.

Scripted REST API security

Configuration	Scripted REST API	Scripted REST Resource		
	Default ACLs	Requires authentication	Requires ACL authorization	ACLs
The resource is public. No authentication or ACL is required.	Any value	False	Any value	Any value
The resource requires basic authentication only. No ACL is required.	Any value	True	False	Any value
The resource requires basic authentication.	No ACL selected	True	True	No ACL selected

Configuration	Scripted REST API		Scripted REST Resource	
	Default ACLs	Requires authentication	Requires ACL authorization	ACLs
on only. ACL is required.				
An ACL selected in the resource record is required.	Any value	True	True	One or more ACLs selected
An ACL selected in the scripted REST API record is required.	One or more ACLs selected	True	True	No ACL selected

Scripted REST API error objects

Scripted REST APIs include error objects that allow you to respond to a request with a standard HTTP error message when an error occurs during request processing. You can use error objects in scripted REST API resources to alert requesting clients of errors. Use error objects to respond to incoming requests, not to catch errors within your server-side code.

Error response format

The content type of the response depends on the request Accept header. If the Accept header specifies an unsupported format, such as image/jpeg, the error response uses JSON.

Error responses follow this format:

```
{  
  "error": {  
    "message": "My error message",  
    "detail": "My details"  
  },  
}
```

```
    "status": "failure"
}
```

The numeric status code, such as 404, is included in the response Status code header, not in the response body.

Automated Test Framework support

The [Test your apps with the ATF](#) (ATF) supports Inbound REST test steps. You can create automated tests for custom Inbound REST APIs that you create. Creating tests for your custom REST APIs simplifies upgrade testing, and makes it possible to verify that modifications to a REST API are backward compatible. See [Administering REST test step configurations](#) and [ATF REST test step configurations](#).

Developer training

In the ServiceNow® Developer Site, you can find training for [Scripted REST APIs](#).

- [Create a scripted REST API](#)

Create a scripted REST API to define web service endpoints.

- [Scripted REST APIs good practices](#)

Follow these guidelines when designing and implementing scripted REST APIs.

- [Scripted REST API examples](#)

Multiple examples are available demonstrating how to create and use scripted REST APIs.

Create a scripted REST API to define web service endpoints.

Before you begin

Role required: web_service_admin

About this task

By default, scripted REST APIs contain an ACL that prohibits users with the snc_external role from making requests to the API.

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Click **New**.
3. Enter a **Name** for the service.
The **API ID** is set automatically based on the **Name**. You can modify the **API ID** if needed.
4. Click **Submit**.

What to do next

After you create the API, configure the service as needed such as by creating resources, assigning ACLs, or specifying supported request and response formats.

- [Create a scripted REST API resource](#)

Create a scripted REST API resource to define the HTTP method, the processing script, and to override settings from the parent service.

- [Define scripted REST API headers](#)

Define scripted REST API headers to control which headers the API accepts and can respond with.

- [Define available query parameters](#)

Define available query parameters to control what values a requesting user can pass in the request URI.

- [Associate query parameters to a resource](#)

Associate scripted REST API query parameters to a resource.

- [Configure a scripted REST API to require an ACL](#)

Requests to scripted REST APIs respect platform ACLs, and the requesting user must meet any table ACL requirements to access instance data. Additionally, you can configure the scripted REST API to require a specific ACL.

- [Enable versioning for a scripted REST API](#)

Enable versioning for a scripted REST API to provide multiple versions of the API while maintaining compatibility with existing integrations.

- [Control request and response content type](#)

Controls which content types are allowed in scripted REST API requests and responses.

- [Controlling maximum request size](#)

You can specify the maximum file size allowed in a scripted REST API request payload.

Create a scripted REST API resource to define the HTTP method, the processing script, and to override settings from the parent service.

Before you begin

There must be a scripted REST API defined before you can create resources.

Role required: web_service_admin

About this task

By default, any new Scripted REST API resource you create contains an ACL that prohibits users with the snc_external role from making requests to the API.

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Select a scripted REST API record.
3. In the **Resources** related list, click **New**.
4. Enter a **Name**.
The resource name affects the URI for sending requests to the API.
5. Select the **HTTP method** this resource implements, such as **GET**.

6. In the **Script** field, define how the operation parses and responds to requests.
7. (Optional) Override settings from the parent REST API as needed, such as by specifying different security settings or valid content types.
8. (Optional) On the **Documentation** tab, provide a **Short description** explaining how to access the resource.
This information appears when exploring this resource using the REST API Explorer.
9. Click **Submit**.

What to do next

After creating the resource, you can associate headers and query parameters. For details, see [Define scripted REST API headers](#) and [Define available query parameters](#).

Related concepts

- [Scripted REST API examples](#)

Define scripted REST API headers to control which headers the API accepts and can respond with.

Before you begin

There must be a scripted REST API defined before you can create headers.

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Select a scripted REST API record.
3. In the **Request Headers** related list, click **New**.
4. Enter a **Header name**.

5. Enter a **Short description** and **Example value** to explain how to use the header.
6. Click **Submit**.
7. (Optional) From the Request Headers tab, locate the header name and set **Is required** to "true" to make this header mandatory for all requests to associated scripted REST resources.

What to do next

After defining available headers, associate the headers with a scripted REST resource.

Define available query parameters to control what values a requesting user can pass in the request URI.

Before you begin

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Select a scripted REST API record.
3. In the **Query Parameters** related list, click **New**.
4. Specify the **Query parameter name**.
5. Enter a **Short description** and **Example value** to explain how to use the parameter.
6. Click **Submit**.
7. (Optional) From the Query Parameters tab, locate the query parameter and set **Is required** to "true" to make this query parameter mandatory for all requests to associated scripted REST resources.

What to do next

After defining available query parameters, associate the parameters to a scripted REST resource. For details, see [Associate query parameters to a resource](#).

Associate scripted REST API query parameters to a resource.

Before you begin

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Select a scripted REST API record.
3. In the **Resources** related list, click **New**
4. In the **API query parameter**, select or enter the query parameter to associate with the resource.
5. In the **API resource**, select or enter the scripted REST API resource to associate with the query parameter.
6. Click **Submit**.

What to do next

After associating the parameters with a scripted REST resource, configure any required ACLs for the API or endpoint. For details, see [Configure a scripted REST API resource to require an ACL](#).

Requests to scripted REST APIs respect platform ACLs, and the requesting user must meet any table ACL requirements to access instance data. Additionally, you can configure the scripted REST API to require a specific ACL.

Before you begin

Role required: web_service_admin

About this task

The ACLs selected in this task apply to all API endpoints.

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Select a scripted REST API.
3. In the **Default ACLs** field, select one or more ACLs that meet the security needs for the API. Select only those ACLs that have a **Type** of **REST_Endpoint**.
A requesting user must satisfy at least one of the selected ACLs. It is not necessary to satisfy all selected ACLs.
4. Click **Update**.

What to do next

You can override the API security settings for each individual API resource/endpoint. For details, see [Configure a scripted REST API resource to require an ACL](#).

- [Configure a scripted REST API resource to require an ACL](#)
By default, API resources/endpoints inherit security settings from the parent API. Define custom ACLs for a specific resource/endpoint to override the inherited settings.

By default, API resources/endpoints inherit security settings from the parent API. Define custom ACLs for a specific resource/endpoint to override the inherited settings.

Before you begin

Role required: web_service_admin or admin

About this task

ACLs are checked for an authenticated user only.

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Select a scripted REST API.
3. In the **Resources** related list, select a resource.
4. In the **Security** tab, select the **Requires authentication** check box.
You must select this check box to require an ACL for the resource. If you clear this check box, the resource becomes public and requires no credentials. Clear this check box only if you want to allow unauthenticated requests to access the resource, even if the parent REST service requires an ACL.
5. Select the **Requires ACL authorization** check box.
6. In the **ACL** field, select one or more ACLs that meet the security needs for the endpoint. Select only those ACLs that have a **Type** of **REST_Endpoint**. Only users who have roles defined in the selected REST_Endpoint type ACL are granted access to this resource.
Selecting an ACL for a resource overrides any ACLs selected for the parent web service. Leave this field blank to use the ACLs selected for the parent web service.

Related tasks

- [Configure a scripted REST API to require an ACL](#)

Enable versioning for a scripted REST API to provide multiple versions of the API while maintaining compatibility with existing integrations.

Before you begin

There must be a scripted REST API defined before you can enable versioning.

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.
2. Select a scripted REST API.

3. Under **Related Links**, click **Enable versioning**.

The **Enable versioning** pop up appears. The **Make version v1 default** check box is selected by default.

4. (Optional) Clear the **Make version v1 default** check box to enable versioning without a default version.

Versioned APIs without a default version are accessible only by using the version-specific URI. Make version v1 default, or select a different version as default after you enable versioning.

5. Click **OK**.

The **Versioning** embedded list is added to the Scripted REST Service form. You can add new versions or control which version is default from this list.

- Add a version to a scripted REST API

Add a new version to a versioned scripted REST API to define new API behavior without impacting older versions.

Add a new version to a versioned scripted REST API to define new API behavior without impacting older versions.

Before you begin

There must be a scripted REST Service that has versioning enabled before you can add a new version.

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > Scripted REST APIs**.

2. Select a scripted REST API.

3. Under **Related Links**, click **Add new version**.

The **Add new version** pop up appears.

4. (Optional) Select **Make this version the default** to configure the REST service to use the new version as the default version.

5. (Optional) In the **Copy existing resources from version** choice list, select an existing version to copy all resources from that version to the new version.

6. Click **OK**.

Controls which content types are allowed in scripted REST API requests and responses.

By default, scripted REST APIs support `application/json`, `application/xml`, and `text/xml`. User-defined custom content types (with `json` or `xml` subtypes) are also supported. For example, `application/vnd.collection+json` and `application/vnd.adobe.xdp+xml` are treated as JSON and XML, respectively.

Important: If the request body format is not of a `json` or `xml` subtype, use only the request body `dataStream` field to access the request body. Using request body `data`, `dataString`, `nextEntry()`, or `hasNext()` with a non-json or non-xml format results in a 500 error response.

Setting defaults

You can set default values for the API using the **Default supported request formats** and **Default supported response formats** fields. These fields define acceptable values users can pass in the Content-Type and Accept request headers, respectively. If a requesting user specifies an Accept or Content-Type header not supported by the API or resource, the instance responds with an HTTP error code of 406 or 415.

You can override these values for each resource using the **Supported request formats** and **Supported response formats** on the Scripted REST Service form.

Note: The **Supported request formats** field appears only for PUT, POST, and PATCH resources.

Using wildcard values

You can use wildcard values when specifying valid content types.

- To perform a single-character wildcard search, use the percent sign (%) character. This wildcard finds words that contain any one character in

place the percent-sign-character. For example, to find words such as text or test, search for: te%t.

- To perform a multiple-character wildcard search, use the asterisk (*) character. This wildcard finds words that contain zero or more characters in place of the asterisk-character. For example, to find words such as planned or placed, search for: pl*d.

Using the `x-www-form-urlencoded` content type

If a REST API or resource accepts the `application/x-www-form-urlencoded` content-type, you can retrieve the urlencoded values provided in the request as a JSON map. You can then supply these urlencoded key-value pairs as query parameters, in the request body, or both. They are combined and stored in the request parameters. Access these parameters through the `request.queryParams` object.

For example, if your API is defined to accept the `application/x-www-form-urlencoded` content-type and your API is implemented as follows,

```
(function process /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    response.setBody(request.queryParams);  
} )(request, response);
```

... then the following request yields the respective response:

```
POST to localhost:8080/api/now/some_api/some_resource?name3=value3&name4=value4
```

Body:

```
name1=value1&name2=value2
```

Response:

```
{ "result":  
    { "name4": [ "value4" ], "name3": [ "value3" ], "name2": [ "value2" ],  
      "name1": [ "value1" ]  
    }  
}
```

Sending binary type in a response

When sending a binary type in a response, you must set the response content type and write the binary stream directly using a `RESTAPIResponseStream` object. You can access this object by calling `getStreamWriter()` on the response object. For more information, see.

Related concepts

- [RESTAPIResponseStream - Scoped, Global](#)

You can specify the maximum file size allowed in a scripted REST API request payload.

The file size limit applies when accessing any of the following variables or functions from a `RESTAPIRequestBody` object.

- `data`
- `dataString`
- `nextEntry()`
- `hasNext()`

Accessing these variables or functions with a request payload that exceeds the maximum size causes the service to respond with error code 400.

The file size limit does not apply when accessing the `dataStream` variable.

Maximum request size properties

Several properties control the maximum allowed request size. Add these properties to the System Properties [sys_properties] table to specify a maximum request size.

Properties

Property	Default value	Description
<code>glide.rest.scripted.max_inbound_content_len_gth_mb</code>	10	The maximum size, in megabytes, for a scripted REST request

Property	Default value	Description
		body that is not zipped.
glide.rest.scripted.max_inbound_gzip_content_length_mb	1	The maximum size, in megabytes, for a scripted REST request body that is zipped.
glide.rest.max_content_length	10	The maximum size, in megabytes, for a scripted REST request body, whether or not it is zipped. Maximum: 25 As a result, even if glide.rest.scripted.max_inbound_content_length_mb or glide.rest.scripted.max_inbound_gzip_content_length_mb are set, the request body is limited to the value of glide.rest.max_content_length.

Follow these guidelines when designing and implementing scripted REST APIs.

Follow REST API conventions

Use REST API standards to provide a consistent and easy to use interface for clients. REST API conventions define specific behavior for each type of method. Use the following guidelines as a starting point for designing your API.

- GET operations only query data. A GET request should never modify data.
- POST operations create new records but do not modify existing records.
- PUT and PATCH operations modify existing records.
- DELETE operations destroy records.

Use versioning to control changes to your API

Use versioning to implement new functionality and avoid breaking existing integrations. When you introduce significant functionality changes to your API, create a new version of the API first. Do not introduce behavior that will break existing integrations in a published version.

Using versioning allows you to implement significant changes to your API without breaking existing clients. You can then release the new version of the API for new clients while allowing existing clients to upgrade at their own pace.

Encourage clients to use a version-specific API, or configure the API without a default version to force clients to specify a version. You can also make new, optional behavior available by adding an optional parameter to an existing version.

Return an informative HTTP status code

Return a status code that informs the requestor of the success or failure of the request. Return an HTTP status code that helps the client understand the result of the request. Use the following guidelines for common status codes.

Common status codes

Status code	Description
200	Indicates that the request was completed successfully.
201	Indicates that a record was created successfully.

Status code	Description
204	Indicates that a record was deleted successfully.
40X (401, 404)	Status codes in the 400 range indicate a client error, such as 400 for invalid request syntax.
50X (500, 503)	Status codes in the 500 range indicate that a server error occurred. The client request may have been valid or invalid, but a problem occurred on the server that prevented it from processing the request.

Return useful error information

Provide the client with enough information in error messages to allow them to understand the problem without having to refer to your API documentation. An error response should include a helpful error message, as well as an error status code.

For example, when a client queries a record that does not exist, you can return the error message "The specified record does not exist. Ensure that a record with the ID of <id value> exists in the application." along with a 404 status code.

The scripted REST API feature includes several preconfigured error objects you can use for commonly-encountered errors, and a customizable ServiceRequest error object you can use when the preconfigured error objects do not meet your needs.

Enforce and test access controls

Enforce existing access controls and require additional access to modify data. In addition to requiring authentication to access the API, require authorization to access data. Use the GlideRecordSecure API in your scripted REST API scripts. This API ensures that access controls defined on the underlying data are applied for the requesting user.

Require additional access controls for operations that modify data. Requests such as PUT, POST, and DELETE should require a higher level of access than GET. Configure these API resources to require a more strict ACL.

Test your access controls, both authentication and authorization, before releasing the API.

Build tests to verify functionality

Build tests that verify your scripted REST web services functionality as part of your development process. Use repeatable tests to ensure that your API functions the way you expect it to. Testing also helps ensure that changes you make do not affect the expected API behavior after you release a version. You can use a REST client application that supports automated testing, such as Postman, to facilitate testing.

Tests should validate the response code, headers, and body content as appropriate for each resource you implement. You can also use tests to validate authentication requirements, and to confirm that errors return useful responses.

Multiple examples are available demonstrating how to create and use scripted REST APIs.

- [Scripted REST API example - script samples](#)

These examples demonstrate how to create various resource scripts for a scripted REST API.

- [Scripted REST API example - streaming vs object serialization](#)

These examples demonstrate how to send a JSON response using streaming and using default object serialization.

- [Scripted REST API example - streaming file attachments](#)

This example demonstrates how to send an image attachment to a requesting user as a binary stream.

These examples demonstrate how to create various resource scripts for a scripted REST API.

Query parameters GET example

This example demonstrates how to get query parameter values from a request.

```
/**  
 * GET - Sample Request API - Query Params  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    var uri = request.uri;  
    var url = request.url;  
    var queryParams = request.queryParams;  
    var customHeader = request.getHeader('X-Custom');  
  
    return {  
        "uri": uri,  
        "url": url,  
        "queryParams": queryParams,  
        "customHeader": customHeader  
    };  
  
})(request, response);
```

Path parameters GET example

This example demonstrates how to get path parameter values from a request.

```
/**  
 * GET - Sample Request API - Path Params  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    var uri = request.uri;  
    var url = request.url;  
    var path = request.pathParams;  
  
    return {  
        "uri": uri,  
        "url": url,  
        "path_params": path,  
        "path.id": path.id  
    };  
})(request, response);
```

```
    };
}) (request, response);
```

Script include GET example

This example demonstrates how to use a script include to provide a response. By using a script include you can reuse common code and maintain readability in the REST service scripts.

```
/***
 * GET - Sample Request API - Script Include
 */
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
    var responseObj = global.SampleDataUtil.getSampleJSON();
    return responseObj;
}) (request, response);
```

String POST example

This example demonstrates how to parse a POST message with a string body and send a response based on the request.

```
/***
 * POST - Sample Request API - dataString
 * sample usage:
 * var requestBody = request.body;
 * var requestString = requestBody.dataString;
 */
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
    var requestBody = request.body;
    var requestString = requestBody.dataString;
    return {"requestString": requestString};
}) (request, response);
```

Binary POST example

This example demonstrates how to parse a POST message with a binary body and send a response based on the request.

```
/**  
 * POST - Sample Request API - Body  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    var body = request.body.data;  
    //do any additional processing on the request body, such as inserting a new record.  
    return {  
        "body.id": body.id  
    };  
) (request, response);
```

Not acceptable error example

This example demonstrates how to respond with a not acceptable error. Use this error type when the request Accept header value is not supported by the web service.

```
/**  
 * Sample Not Acceptable Error Sample  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    response.setError(new sn_ws_err.NotAcceptableError('sample error message'));  
) (request, response);
```

Bad request error example

This example demonstrates how to respond with a bad request error. Use this error type to indicate a mistake in the request syntax.

```
/**  
 * Bad Request Error Sample  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    response.setError(new sn_ws_err.BadRequestError('sample error message'));  
) (request, response);
```

Conflict error example

This example demonstrates how to respond with a conflict error. Use this error type in the event of multiple conflicting requests, such as multiple updates to the same record.

```
/**  
 * Error Response: Conflict Error Sample  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    response.setError(new sn_ws_err.ConflictError('sample  
error message'));  
})(request, response);
```

Not found error example

This example demonstrates how to respond with a not found error. Use this error type if the requested resource does not exist or is unavailable.

```
/**  
 * Error Response: Not Found Error Sample  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    response.setError(new sn_ws_err.NotFoundError('sample  
error message'));  
})(request, response);
```

Unsupported media type error example

This example demonstrates how to respond with an unsupported media type error. Use this error type to indicate that the Content-Type of the request is unsupported.

```
/**  
 * Error Response: Unsupported Media Type Error Sample  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    response.setError(new sn_ws_err.UnsupportedMediaTypeError('sample  
error message'));  
})(request, response);
```

Service error example

This example demonstrates how to respond with a generic service error. The ServiceError object allows you to define the status code, message, and error detail. Use a ServiceError if the predefined error types do not meet your needs.

```
/**  
 * Error Response: Custom Error Sample  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
    var myError = new sn_ws_err.ServiceError();  
    myError.setStatus(418);  
    myError.setMessage("I am a Teapot");  
    myError.setDetail("Here are the details about this error");  
    response.setError(myError);  
})(request, response);
```

Scripted REST resource script example

This sample REST API resource script parses the name and id values from the request body and returns those values in the response.

```
/**  
 * POST - Sample Request API - Body  
 */  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
var body = request.body.data,  
id0, name0, id1, name1;  
name0 = body[0].name; // 'user0'  
id0 = body[0].id; // '1234'  
name1 = body[1].name; // 'user1'  
id1 = body[1].id; // '5678'  
  
return {  
    "id": id0,  
    "name": name0,  
    "id1": id1,  
    "name1": name1  
};  
})(request, response);
```

Requests

The API can accept both XML and JSON requests.

Requests

JSON Request	XML Request
<pre>POST /api/sn_demo_api/v1/example/body HTTP/1.1 Content-Type: application/json Accept: application/json Host: <instance>.service-now.com Connection: close Content-Length: 91 [{ "name": "user0", "id": 1234 }, { "name": "user1", "id": 5678 }]</pre>	<pre>POST /api/sn_demo_api/v1/example/body HTTP/1.1 Content-Type: application/xml Accept: application/json Host: <instance>.service-now.com Connection: close Content-Length: 152 <request><entry> <name>user0</name> <id>1234</id> </entry> <entry> <name>user1</name> <id>5678</id> </entry> </request></pre>

Responses

Both requests specify application/json as the Accept header value. This causes either response to use JSON formatting, even if the request content type is XML.

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 04 Aug 2015 15:20:44 GMT
Server: ServiceNow
Connection: close
Set-Cookie: BIGipServerpool_<Instance>=880838154.47166.000
0; path=/
```

```
{"result": {"id": 1234, "id1": 5678, "name": "user0", "name1": "user1"}}
```

Related concepts

- RESTAPIRequest - Scoped, Global
- RESTAPIRequestBody - Scoped, Global
- RESTAPIResponse - Scoped, Global
- RESTAPIResponseStream - Scoped, Global

These examples demonstrate how to send a JSON response using streaming and using default object serialization.

Streaming vs object serialization

When sending a response, you can send a response as a stream or serialize an object. There are advantages and disadvantages to either approach. Pick a technique based on the needs of your integration.

Generally, if the response object is simple, can be represented as XML or JSON, and is a consistent size, use object serialization. If using a format other than XML or JSON, or if the size of the response varies, use streaming.

Streaming the response

Using a streaming responses provides advantages in response time, instance performance, and content flexibility, but adds additional complexity to the script. When using streaming, you are responsible for formatting the response, setting the response status, and setting the Content-Type header. When streaming a response, the requesting user receives a response quickly because the entire response does not need to be created before starting streaming.

This example demonstrates a Scripted REST Resource script that returns an array of incident records using streaming.

Example

```
/**  
 * Sample Scripted REST Resource that returns custom JSO
```

```
N objects with properties from Incident GlideRecords
 * This sample uses ServiceNow JavaScript API to query incident records
 * and then iterates over those records to build and stream a custom JSON object that
 * includes some values from the incidents
 */
(function runOperation(/*RESTServiceRequest*/ request, /*RESTServiceResult*/ response) {
    var writer = response.getWriter(),
        hdrs = {},
        table = 'incident',
        record_limit = 100,
        gr = new GlideRecord(table);

    hdrs['Content-Type'] = 'application/json';
    response.setStatus(200);
    response.setHeaders(hdrs);

    gr.setLimit(record_limit);
    gr.query();

    // start building response object
    writer.writeString("{\"results\":[");

    // iterate over incident records and build JSON representations to be streamed out.
    while (gr.next()) {
        var incidentObj = {};

        incidentObj.number = gr.number + '';
        incidentObj.short_description = gr.short_description + '';
        writer.writeString(global.JSON.stringify(incidentObj));

        if (gr.hasNext()) {
            writer.writeString(",");
        }
    }

    // close the response object
});
```

```
    writer.writeString("]}");
})(request, response);
```

A request to this resource returns the following response.

```
// sample response
/*
HTTP/1.1 200 OK
Content-Type: application/json
Server: ServiceNow

// sample response number of records returned has been truncated for this example

{
  "results": [
    {
      "number": "INC0011301",
      "short_description": "lorem ipsum short description 0 my new incident"
    },
    {
      "number": "INC0011302",
      "short_description": "lorem ipsum short description 1 my new incident"
    },
    {
      "number": "INC0011303",
      "short_description": "lorem ipsum short description 2 my new incident"
    },
    {
      "number": "INC0011304",
      "short_description": "lorem ipsum short description 3 my new incident"
    },
    {
      "number": "INC0011309",
      "short_description": "lorem ipsum short description 8 my new incident"
    },
    {
      "
```

```
    "number": "INC0011310",
    "short_description": "lorem ipsum short description 9 my
new incident"
},
{
    "number": "INC0011311",
    "short_description": "lorem ipsum short description 0 my
new incident"
},
{
    "number": "INC0011312",
    "short_description": "lorem ipsum short description 1 my
new incident"
},
{
    "number": "INC0011313",
    "short_description": "lorem ipsum short description 2 my
new incident"
},
{
    "number": "INC0011314",
    "short_description": "lorem ipsum short description 3 my
new incident"
},
{
    "number": "INC0011315",
    "short_description": "lorem ipsum short description 4 my
new incident"
},
{
    "number": "INC0011326",
    "short_description": "lorem ipsum short description 15 m
y new incident"
}
]
}
*/

```

Building an object

Using object serialization allows you to take advantage of ServiceNow provided serialization and content negotiation. When serializing an object instead of streaming, the entire object must be created and serialized before the client receives a response. This may delay the

response, or require a large amount of system resources if the response object is very large. Object serialization is available only for XML or JSON responses. Responses using a different format must use streaming.

This example returns the same Incident data as the streaming example, but collects all of the response data in an array before sending the response.

Example

```
/***
 * Sample Scripted REST Resource returns an array of custom JSON objects that include 2 incident properties.
 * This sample uses ServiceNow JavaScript API to query incident records and then iterates
 * over those records building a custom JSON object that includes 2 values from the incident records.
 * note that because we are returning a simple JSON object we can rely on built in serialization
 * to set the content-type header as well as response status. The 'result_arr' object will not be returned
 * until it has been completely built and stored
 */
(function runOperation(/*RESTServiceRequest*/ request, /*RESTServiceResult*/ response) {
    var table = 'incident',
        record_limit = 100,
        result_arr = [],
        gr = new GlideRecord(table);

    gr.setLimit(record_limit);
    gr.query();

    // iterate over incident records and build JSON representations to be streamed out.
    while (gr.next()) {
        var incidentObj = {};

        incidentObj.number = gr.number + '';
        incidentObj.short_description = gr.short_description + '';
```

```
        result_arr.push(incidentObj);
    }

    return result_arr;
}) (request, response);
```

A request to this resource returns the following response.

```
/*
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Server: ServiceNow

// sample response number of records returned has been truncated for this example

{
  "result": [
    {
      "short_description": "lorem ipsum short description 0 my new incident",
      "number": "INC0011301"
    },
    {
      "short_description": "lorem ipsum short description 1 my new incident",
      "number": "INC0011302"
    },
    {
      "short_description": "lorem ipsum short description 2 my new incident",
      "number": "INC0011303"
    },
    {
      "short_description": "lorem ipsum short description 3 my new incident",
      "number": "INC0011304"
    },
    {
      "short_description": "lorem ipsum short description 4 my new incident",
      "number": "INC0011305"
    },
    {
      "short_description": "lorem ipsum short description 5 my new incident",
      "number": "INC0011306"
    }
  ]
}
```

```
    "short_description": "lorem ipsum short description 6 my  
new incident",  
    "number": "INC0011307"  
}, {  
    "short_description": "lorem ipsum short description 7 my  
new incident",  
    "number": "INC0011308"  
}, {  
    "short_description": "lorem ipsum short description 8 my  
new incident",  
    "number": "INC0011309"  
}, {  
    "short_description": "lorem ipsum short description 9 my  
new incident",  
    "number": "INC0011310"  
}  
}  
*/
```

Related concepts

- RESTAPIRequest - Scoped, Global
- RESTAPIRequestBody - Scoped, Global
- RESTAPIResponse - Scoped, Global
- RESTAPIResponseStream - Scoped, Global

This example demonstrates how to send an image attachment to a requesting user as a binary stream.

Example

```
/**  
 * Sample Scripted REST Resource that returns a stream o  
f binary representing an attachment  
* This sample uses ServiceNow JavaScript API GlideSysAtt  
achmentInputStream to get an attachment as a stream then  
* users WriteStream to stream the response.  
*/  
(function process(/*RESTAPIRequest*/ request, /*RESTAPIRes  
ponse*/ response) {
```

```
var hdrs = {},  
    attachment_sys_id = '1852fd52471321009db4b5b08b9a7  
1a9';  
  
hdrs['Content-Type'] = 'image/jpeg';  
response.setStatus(200);  
response.setHeaders(hdrs);  
  
var writer = response.getWriter();  
var attachmentStream = new GlideSysAttachmentInputStream(attachment_sys_id);  
writer.writeStream(attachmentStream);  
})(request, response);
```

A request to this resource returns the following response.

```
// sample response  
/*  
HTTP/1.1 200 OK  
Set-Cookie: glide_session_store=SYSTEM; Expires=Fri, 30-Oct-2015 21:57:00 GMT; Path=/; HttpOnly  
Content-Type: image/jpeg  
Transfer-Encoding: chunked  
Date: Fri, 30 Oct 2015 21:26:59 GMT  
Connection: close  
Server: ServiceNow  
  
<binary response body excluded from this sample>  
*/
```

Related concepts

- RESTAPIRequest - Scoped, Global
- RESTAPIRequestBody - Scoped, Global
- RESTAPIResponse - Scoped, Global
- RESTAPIResponseStream - Scoped, Global

SOAP web service

Simple Object Access Protocol (SOAP) is an XML-based protocol for accessing web services over HTTP.

You can use SOAP to access data on your instance. Available SOAP web services are WS-I compliant, as outlined in the WS-I Basic Profile 1.0.

Web service provider

ServiceNow publishes its underlying table structures and associated data using the following web service methods:

- **Direct web services:** Use a URL query to request a table's WSDL.
- **SOAP web service import sets:** Use import tables and transform maps to automate web service requests for tables.
- **Scripted SOAP web services:** Use custom JavaScript to execute SOAP web services requests.

Note: SOAP messages are sent with the assumption that the recipient is XML compliant. No encoding is applied to a SOAP message. SOAP always decodes responses as UTF-8, the XML encoding header is not used.

WSDL

All tables and import sets dynamically generate Web Service Definition Language (WSDL) XML documents that describe its table schema and available operations.

You can obtain a table's WSDL by issuing a URL call to your instance that contains the name of the table and the **WSDL** parameter. For example:

```
https://myinstance.service-now.com/incident.do?WSDL
```

All dynamically generated and served ServiceNow WSDLs accessible via HTTP are available for use under the terms defined in the Open Source Initiative OSI - Apache License, Version 2.0 license agreement.

Long-running SOAP request support

The Now Platform supports long-running SOAP requests by preventing socket timeouts due to inactivity of the network connection while the requests are in process.

This functionality improves the efficiency of the ODBC driver when requesting large numbers of records, doing aggregate queries, or using order by expressions that require sorting.

By default, the system provides timeout protection for web services clients provided by ServiceNow such as the ODBC driver and the MID Server. You can add timeout protection to your custom web services with system properties.

Timeout protection

Web services clients receive a 307-Temporary Redirect to keep long sessions alive and prevent a timeout due to socket inactivity. A 307-Temporary Redirect causes web services clients which support the status code to repeat their last request to the location specified in the HTTP location header. The value of the location header is the same URL that the web services client originally specified. The use of 307-Temporary Redirects is WS-I compliant.

A web service request that exceeds the timeout limit specified in `glide.soap.request_processing_timeout` can only receive a 307-Temporary Redirect when all of these conditions are met:

- The value of `glide.soapprocessor.allow_long_running_threads` is true.
- The request includes a `redirectSupported=true` URL parameter.
- The request is session-aware (supports HTTP cookies).
- The number of redirects has not exceeded the value set by `glide.soap.max_redirects`.

If any of these conditions is not met, the web service client receives a 408 Request Timeout error.

Note: To ensure that applications experience a socket timeout rather than a 408 Request Timeout, set the `glide.soap.request_processing_timeout` property to a value larger than the shortest socket timeout setting in effect for the connection between the application and the instance (300 seconds for hosted instances).

SOAP web services security

An instance enforces web service security using a combination of basic authentication challenge/response over the HTTPS protocol and system-level access control lists (ACLs) using contextual security. Administrators can control what system resources web services users can access by granting them one of the SOAP roles.

SOAP roles

To use SOAP web services, you must have the appropriate role for the operation you want to perform. Also, you must have any other roles required to access the target tables.

SOAP Roles

Role	Description
soap	Can perform all SOAP operations.
soap_create	Can insert new records.
soap_delete	Can delete existing records.
soap_ecc	Can query, insert, and delete records on the Queues [ecc_queue] table.
soap_query	Can query record information.
soap_query_update	Can query record information and update records.

Role	Description
soap_script	Can run scripts that specify a .do endpoint. This role is required for running scripted web services.
soap_update	Can update records.
import_admin	Can manage all aspects of import sets and imports. Required for access to the Import Set Row [sys_import_set_row] table.
import_transformer	Can manage import set transform maps and run transforms. Required for access to the Import Set Row [sys_import_set_row] table.

Default web services role requirements

By default, a set of processor ACL rules require users to have the soap_query role to make WSDL, XSD, and XML schema requests.

If you want to change these role requirements, you can deactivate the ACL rules.

Web service processor ACLs

Web service processor ACLs						
Access Controls		New	Go to	Name	Search	Navigation
All > Type = processor > Updated between May 21, 2013 00:00 and Jun 4, 2013 23:59						
Actions	Name	Operation	Type	Active	Updated by	Updated
<input type="checkbox"/>	SchemaProcessor	execute	processor	true	admin	May 22, 2013 00:16
<input type="checkbox"/>	WSDLProcessor	execute	processor	true	admin	May 22, 2013 00:16
<input type="checkbox"/>	XSDProcessor	execute	processor	true	admin	May 22, 2013 00:17
<input type="checkbox"/> Actions on selected rows...						

Basic authentication

To enforce basic authentication for the user associated with the instance for each WSDL or SOAP message request, administrators can set the property `glide.basicauth.required` to **true**.

When enabled, each WSDL and SOAP request must contain an "Authorization" header as specified in the [Basic Authentication](#) protocol.

Because web services requests are non-interactive, the **Authorization** header is always required during a request.

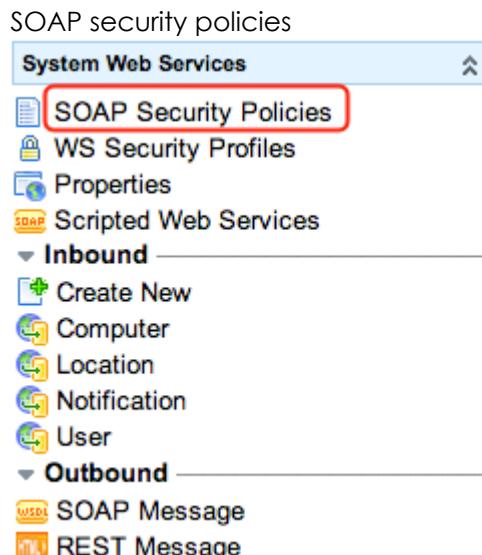
Note: If configured, basic authentication refers to local credentials or LDAP authentication.

Supplying basic authentication information with every request (whether or not it is required) has the added advantage that the user supplied in the basic authentication credentials can be associated web service invocation. For example, when creating an Incident record, the journal field lists the user ID contained in the basic authentication header instead of the default guest user.

SOAP security policies

The Enhanced Web Service Provider - Common plugin adds the SOAP Security Policies module to the System Web Services application. This module allows administrators to set the following security policies:

- Enable or disable signing SOAP requests when consuming an external web service
- Specify the authentication requirements SOAP requests must meet when communicating over WS-Security.



Certificates required for signed SOAP requests

To sign SOAP requests for WS-Security communications, the following certificates are required:

- X.509 certificate from the requester
- X.509 CA certificate of the certificate authority who signed the requester's certificate

SOAP default security policy

Administrators can specify the SOAP security policy an instance uses with the system property `glide.soap.default_security_policy`. The `glide.soap.default_security_policy` system property specifies the name of the SOAP security policy the instance uses when enforcing Web Services-Security (WSS) for inbound requests.

SOAP default security policy settings

Field	Description
Type	String
Default value	Default Security Policy
Location	Add a system property to the System Properties [sys_properties] table

WS-Security

You can validate signed web services requests using WS-security. Enable WS-Security to:

- Verify that SOAP messages originate from a known sender
- Verify that SOAP messages have not been altered in transit

ServiceNow supports [WS-Security 1.1](#) to validate signed web services requests.

Note: WS-Security is not used as an encryption mechanism, HTTPS protocol is used to encrypt all communications.

WS-Security is intended to work with basic authentication. When an instance receives a SOAP message, it reviews the basic authentication header to determine if the SOAP user has rights to the instance. It reviews the WS-Security header to determine the validity of the incoming message. Requests affected by attacks, such as a man-in-the-middle attack, have an invalid WS-Security header and are blocked.

WS-Security profiles

A WS-security profile determines how a web services message is authenticated when WS-security is enabled. The following mechanisms can be used to authenticate web services requests:

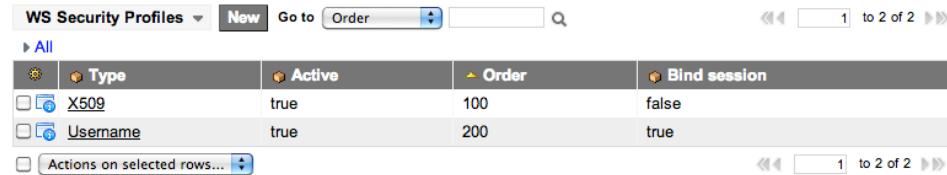
Web service authentication mechanisms

Authentication mechanism	Description
Certificate verification	Verifies the certificate associated with the request. Verifying the request's certificate requires uploading the requester's certificate and certificate authority.
User credentials	Authenticates the web services request by verifying the user credentials associated with the request. This type of authentication can either verify that the request's credentials match an existing user's credentials or that the request's credentials match a user name and password provided in the profile record.

Specify the authentication mechanism you want to use when you [create a new WS-security profile](#).

The WS-Security Profiles module lists the WS-Security profiles that are currently in effect.

WS-Security Profiles module



Type	Active	Order	Bind session
X509	true	100	false
Username	true	200	true

WS-Security error logging

The `glide.processor.debug.SOAPPProcessor` system property allows error messages about WS-security to be displayed in the transaction log.

The system property `glide.processor.debug.SOAPPProcessor` enables (true) or disables (false) debugging messages for SOAP processing such as certificate and keystore checks.

glide.processor.debug.SOAPPProcessor fields

Field	Description
Type	true false
Default value	false
Location	Add a system property to the System Properties [sys_properties] table

WSS X.509 Token Profile

Use the X.509 framework for a WSS X.509 security profile. An X.509 certificate is used to validate a public key that is then used to sign the incoming SOAP message. It specifies a binding between a public key and a set of attributes that includes at least the following:

- subject name
- issuer name
- serial number

- validity interval

Use the [X.509 authentication framework](#) as defined by the [Web Services Security: SOAP Message Security specification](#).

Upload the certificate and reference it in the **X509 Certificate** field. If a bound session, select the user to impersonate when the WS-Security authentication succeeds.

WSS X.509 Security Profile

The screenshot shows a configuration interface for a 'WS Security Profile'. At the top, there's a header bar with a back arrow, a title 'WS Security Profile' with a required field indicator, and buttons for 'Update', 'Delete', and navigation. Below the header, there are four input fields: 'Type' set to 'X509', 'Bind session' with a checked checkbox, 'Run as user' set to 'ITIL User', and 'X509 Certificate' set to 'WSS'. There are also search and refresh icons next to the certificate field. At the bottom, there's an 'Order' field set to '100'.

WSS UsernameToken Profile

When specifying the X.509 Token Profile, you can also supply a UsernameToken in the SOAP request.

A UsernameToken is used as a means of identifying the requester by "user name", and optionally using a password, shared secret, or password equivalent, to authenticate that identity.

There are two ways to authenticate a UsernameToken.

1. Authenticate with existing user credentials.

Authenticate with existing user credentials

The screenshot shows a configuration interface for a 'WS Security Profile'. The 'Type' field is set to 'Username'. Under 'Bind session', the 'Profile action' dropdown is set to 'Authenticate with user' and the 'User field to match UserName' dropdown is set to 'Email'. Other fields include 'Bind session' (checkbox checked), 'Order' (set to '100'), and 'Profile action' (dropdown). There are also search and refresh icons.

Use the user name of the incoming SOAP request to look up a user by the specified **User** field to match the **UserName** value. The system uses the password value in the incoming UsernameToken to authenticate the request. When the **Bind session** option is selected, the user that authenticates successfully is used for the session.

2. Authenticate with specified user credentials.

Authenticate with specified user credentials

The screenshot shows a configuration interface for a 'WS Security Profile'. At the top, there's a blue header bar with the title 'WS Security Profile' and several action buttons: 'Update', 'Delete', and icons for copy, paste, and refresh. Below the header, the form has two columns of fields. The left column contains 'Type' (set to 'Username'), 'Run as user:' (set to 'ITIL User'), and 'Order' (set to '100'). The right column contains 'Bind session:' (with a checked checkbox), 'Profile action:' (set to 'Specify user to authenticate'), 'User name:' (set to 'ws_user'), and 'User password:' (with a masked value). There are also small search and refresh icons between the two columns.

Authenticate using login credentials unrelated to users in the User table. When the **Bind session** option is selected, the user that is specified in the **Run as user** field is used for the session.

Note: The [UsernameToken Profile](#) cannot be used independent of the X.509 Token Profile.

Strict security for web services

By default, basic authentication for web services only determines whether a user is authorized to access the instance with a SOAP connection. Once authorized, any user can access any table published as a web service.

The system property **Enforce strict security on incoming SOAP requests** changes this behavior and requires that users meet [Contextual Security Manager](#) requirements to access instance resources from web services.

With this property enabled, only users that have the proper SOAP role and also meet the ACL conditions the table and operation can perform that operation from a SOAP connection.

Mutual authentication for web services

Mutual authentication is supported for outbound web services.

SOAP session management and reporting

A SOAP session is a Glide session established with an instance by any external SOAP client, such as a web services client application, a ServiceNow MID Server, or the ServiceNow ODBC driver. SOAP sessions are included in the list of user sessions at **User Administration > Logged in users**. The ?SOAP URLs identify SOAP sessions.

SOAP session properties

Certain properties control how SOAP sessions are maintained.

SOAP session properties

Property	Description
glide.soap.invalidate_session_timeout	<p>Duration, in seconds, that an active session remains open. After this duration is reached, the instance deactivates the session and reclaims any system resources. If the client sends another request after the timeout duration is reached, the instance establishes a new session.</p> <p>This property accepts values from 5 to 1200 seconds (20 minutes).</p> <ul style="list-style-type: none">• Type: integer• Default value: 60• Location: Add to the System Properties [sys_properties] table

Note: To learn more about properties that affect SOAP web services processing, see the following topics in Instance Security Hardening Settings:

- Access control [Access control \(instance security hardening\)](#)

- Basic auth: [SOAP requests](#)

- [Scripted SOAP web services](#)

Scripted SOAP web services allow a ServiceNow administrator to create custom SOAP web services.

- [Direct web services](#)

A direct web service is available for any table in the system if the correct access control list is configured.

- [SOAP web service import sets](#)

Web service import sets complement direct web services and scripted SOAP web services by providing a web service interface to import sets tables.

- [AttachmentCreator SOAP web service](#)

Attach documents to records in ServiceNow by sending a SOAP message targeting the ecc_queue table.

- [Override a SOAP endpoint](#)

The SOAP endpoint address where the SOAP message is posted is consistent with the endpoint of the WSDL.

- [Enable HTTP compression](#)

By default, the SOAP request is accepted un-compressed and the result of the request is returned un-compressed.

- [Prevent empty elements in SOAP messages](#)

By default, an instance does not omit empty elements, elements with NULL or NIL values, from SOAP messages.

- [Insert related records using SOAP](#)

Support is available for inserting hierarchical data into tables or web service import set tables. The hierarchical data in the Insert API is automatically mapped to related records of the targeted table.

- [Specify requirement for signed SOAP requests](#)

Use a SOAP security policy to specify whether the instance requires signed SOAP requests for all inbound SOAP traffic.

- [Activate the Enhanced Web Service Provider - Common plugin](#)

Administrators can activate the Enhanced Web Service Provider - Common plugin to enable unsigned WS-Security requests and specify what authentication requirements SOAP requests have.

- [Configure SOAP security](#)

Administrators can configure web service security for inbound SOAP requests made to the ServiceNow instance.

- [Set the SOAP default security policy](#)

Set the SOAP default security policy.

- [Create a new security policy](#)

Administrators can specify which security profiles WS-Security communications must meet by creating a new security policy.

- [Create a new WS-Security profile](#)

Create a new WS Security profile to define how to authenticate a web services message when WS-Security is enabled.

- [Enforce strict security for inbound SOAP](#)

Strict security for web services requires that users meet Contextual Security requirements to access instance resources.

- [Enable WS-Security verification](#)

Administrators can enable Web Services Security (WSS) verification from the Web Services system properties.

- [Debug incoming SOAP envelope](#)

To capture incoming SOAP envelope XML in the system log, add the property `glide.processor.debug.SOAPProcessor` with a value of `true`.

- [View a SOAP session log](#)

You can view a user's log from a SOAP session.

- [Basic authentication code samples](#)

Samples of basic authentication code for several programming languages and versions.

- [Example WS-Security SOAP envelope header](#)

An example of a valid WS-Security SOAP envelope header.

- [WS-Security properties](#)

These properties control the behavior of WS-Security X.509 tokens.

- [WS-Security error messages](#)

An instances produces one of the following error messages when it encounters an issue with a WS-security SOAP message.

- [LongRunningSOAPRequestProps](#)

The following properties are available for long-running SOAP requests.

Scripted SOAP web services

Scripted SOAP web services allow a ServiceNow administrator to create custom SOAP web services.

You can define input and output parameters for the SOAP web service and use JavaScript to perform operations. Though this feature is very powerful, use [direct web services](#) or [SOAP web service import sets](#) whenever possible since they are simpler to implement and maintain.

Security

Scripted SOAP web services have the same base security options as all SOAP web services. For details on SOAP web services security, see [SOAP web services security](#).

When [strict security](#) is enforced on a system, the HTTP authenticated user must have the [soap_script](#) role to execute the scripted web service.

WSDL

All ServiceNow tables and import sets dynamically generate Web Service Definition Language (WSDL) XML documents that describe its table schema and available operations.

Enforcing WSDL compliance

You can force the response to list output values in the same order as defined in the WSDL.

When you create a scripted SOAP web service, the generated WSDL is based on the Input Parameters and Output Parameters related lists. The actual SOAP response sent by the scripted service is determined by the **Script**. This behavior can cause the script to return output values in a different order than defined in the WSDL.

To enforce the order of output parameters as defined in the related list, select the **WSDL Compliance** check box. When this check box is selected, the web service reorders the parameters returned by the script to match the order in the WSDL.

Note: If additional response parameters are returned by the script, but are not defined in the Response Parameters related list, those parameters are excluded from the response when **WSDL Compliance** is selected.

Example

Output Parameters related list

Parameter	Order
Param 1	200
Param 2	300
Param 3	100

The following is the script that sets values for the defined output parameters. Note that in this example script the parameters are set in a different order than defined in the Output Parameters related list. Also note the additional parameter param4 that is not defined in the related list.

```
Response.param1 = 1;  
Response.param4 = 4;  
Response.param3 = 3;
```

When the **WSDL Compliance** check box is **false**, the SOAP response generated by the script is the following:

```
<response>
    <param1>1</param1>
    <param4>4</param1>
    <param3>3</param1>
</response>
```

When the **WSDL Compliance** check box is **true**, the SOAP response generated by the script is the following:

```
<response>
    <param3>3</param1>
    <param1>1</param1>
</response>
```

Static WSDL

Some web service clients require SOAP access to your instance through a specific WSDL format. This required format may differ from the standard ServiceNow WSDL format. In these cases you can create a static WSDL that matches the required format.

Global variables

To facilitate custom processing of incoming SOAP requests, the following global variables are available in the script context:

- soapRequestDocument: Java org.w3c.dom.Document object representing the incoming SOAP envelope.
- soapRequestXML: String object representing the incoming SOAP envelope XML.
- request: Javascript object that contains mapped values (mapped to input parameter names) of the incoming SOAP envelope.
- response: Javascript object that allows you to customize the response values. See [Customize Response](#)
- [Create a new scripted SOAP web service](#)

Follow these examples to create a new scripted SOAP web service.

- [Customize response](#)

Follow this example to customize and control the XML payload of a SOAP response.

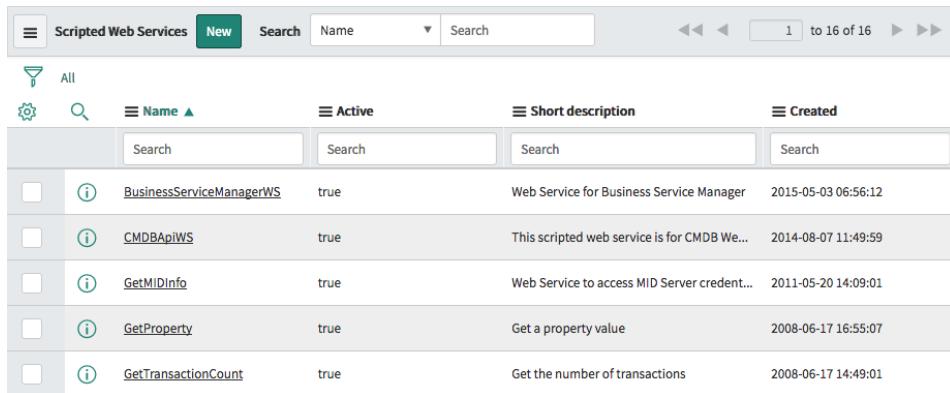
- [Create a scripted SOAP web service using a static WSDL](#)

Follow these examples to create a scripted SOAP web service using a static WSDL.

Follow these examples to create a new scripted SOAP web service.

When the Web Services Provider - Scripted plugin is activated, a new module Scripted Web Services is available under the System Web Services application.

Scripted SOAP web services



The screenshot shows a list view of Scripted Web Services. The top navigation bar includes 'Scripted Web Services', 'New', 'Search', and search fields for 'Name' and 'Search'. Below the header are filters for 'All', 'Name' (sorted ascending), 'Active', 'Short description', and 'Created'. The main table lists five entries:

	Name	Active	Short description	Created
<input type="checkbox"/>	BusinessServiceManagerWS	true	Web Service for Business Service Manager	2015-05-03 06:56:12
<input type="checkbox"/>	CMDBApiWS	true	This scripted web service is for CMDB We...	2014-08-07 11:49:59
<input type="checkbox"/>	GetMIDInfo	true	Web Service to access MID Server credential...	2011-05-20 14:09:01
<input type="checkbox"/>	GetProperty	true	Get a property value	2008-06-17 16:55:07
<input type="checkbox"/>	GetTransactionCount	true	Get the number of transactions	2008-06-17 14:49:01

Example 1: Retrieving a system property

The first step is to define the incoming and return parameters. This is done by adding an entry to the Input Parameters and Output Parameters. These parameters are used to construct and present a meaningful WSDL, and they do not add to the functionality of processing the actual Web Service itself.

GetProperty Input & Output Parameters

The image contains two separate screenshots of the ServiceNow web interface, both titled "GetProperty".

The top screenshot shows the "Input Parameters (1)" tab selected. It displays a single parameter named "property" with an order of 100. The bottom screenshot shows the "Output Parameters (1)" tab selected, also displaying a single parameter named "property" with an order of 100.

Input Parameters (1)	Output Parameters (1)
Web service = GetProperty	Web service = GetProperty
Name	Name
property	property
100	100

The parameters are referenced in the script of the Web Service. Any of the input parameters are retrieved using the following syntax:

```
var a= request.property;
```

The output parameters are set by using the following syntax:

```
response.property="ABC";
```

The following example demonstrates how to retrieve a system property and return it as part of the SOAP response. The example shows how to create a custom scripted web service to do something specific that the base ServiceNow system direct Web Services cannot.

GetProperty web service

The screenshot shows the configuration interface for a Scripted Web Service named 'GetProperty'. The 'Name' field is set to 'GetProperty', 'Application' is 'Global', and 'Function name' is 'execute'. The 'Active' checkbox is checked. The 'WSDL Compliance' checkbox is unchecked. The 'WSDL' field contains the URL <https://demonightlyinterfaces.service-now.com/GetProperty.do?WSDL>. The 'Short description' is 'Get a property value'. A note below states: 'This script reads the parameters in the request and creates the response. For more information about it, see the Wiki and the tutorial. See also the article about the recommended form of the script.' The 'Script' tab is selected, showing the following code:

```
1 * ****
2 * Use the following business rule to invoke this example service
3 *
4 * // create the soap document
5 * var soapdoc = new SOAPEnvelope("GetProperty", "http://www.service-now.com/");
6 * soapdoc.setFunctionName("execute");
7 * soapdoc.addFunctionParameter("property", "glide.db.name");
8 *
9 * // post the request
10 * var soapRequest = new SOAPRequest("http://localhost:8080/glide
/GetProperty.do?SOAP");
11 * var soapResponse = soapRequest.post(soapdoc);
12 * var property = gs.getXMLText(soapResponse, "//executeResponse/property");
13 *
14 * gs.log(property);
15 *
16 * ****
17 *
18 response.property = gs.getProperty(request.property);
```

Example 2: Ordering a Blackberry

Direct web services operate on tables and their data. The following example shows how to initiate a business solution, such as ordering a Blackberry, by invoking a scripted web service. The following input and output parameters support the Blackberry example:

Input output Blackberry

The image contains two screenshots of the ServiceNow parameter configuration interface.

Top Screenshot: Shows the "Input Parameters (2)" tab selected. It lists two parameters: "phone_number" and "requested_for", both with a value of 100. The "Name" column is sorted by order (asc).

Name	Value
phone_number	100
requested_for	100

Bottom Screenshot: Shows the "Output Parameters (1)" tab selected. It lists one parameter: "request_number", with a value of 100. The "Name" column is sorted by order (asc).

Name	Value
request_number	100

This script shows how to use the above parameters to add a Blackberry to the service catalog shopping cart and order it. The request number is returned in the request_number field of the SOAP response.

```
var cart = new Cart();
var item = cart.addItem('e2132865c0a8016500108d9cee411699');
cart.setVariable(item,'original', request.phone_number);

// set the requested for
var gr = new GlideRecord("sys_user");
gr.addQuery("user_name", request.requested_for);
gr.query();

if(gr.next()){
    var cartGR = cart.getCart();
    cartGR.requested_for = gr.getUniqueValue();
    cartGR.update();
}

var rc = cart.placeOrder();
response.request_number= rc.getValue('number');
```

Follow this example to customize and control the XML payload of a SOAP response.

Before you begin

Role required: web_service_admin or admin

Procedure

1. Create a customized XML document using the [XMLDocument](#) script include object.

Note: When creating a scripted web service in a scoped application you must use the [XMLDocument2](#) API.

2. Set its document element to the variable `response.soapResponseElement` in a scripted web service.

For example, the following scripted web service script:

```
var xmldoc = new XMLDocument2();
    xmldoc.parseXML("<myResponse></myResponse>");
    xmldoc.createElementWithTextValue("element_one", "test");
    xmldoc.createElementWithTextValue("element_two", "new2 value");

    var el = xmldoc.createElement("element_three");
    xmldoc.setCurrentElement(el);
    xmldoc.createElementWithTextValue("newChild", "test child element");

    response.soapResponseElement = xmldoc.getDocumentElement();
```

Is used to accept the following request:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tes="http://www.service-now.com/TestCustomResponse">
    <soapenv:Header/>
    <soapenv:Body>
        <tes:execute/>
```

```
</soapenv:Body>  
</soapenv:Envelope>
```

Which will respond with the following SOAP response:

```
<soapenv:Envelope  
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:tes="http://www.service-now.com/TestCustomResponse">  
    <soapenv:Header/>  
    <soapenv:Body>  
        <myResponse>  
            <element_one>test</element_one>  
            <element_two>new2 value</element_two>  
            <element_three>  
                <newChild>test child element</newChild>  
            </element_three>  
        </myResponse>  
    </soapenv:Body>  
</soapenv:Envelope>
```

WSDL support will need to be created externally. The SOAP endpoint will need to be referred back to the scripted web service in question.

Follow these examples to create a scripted SOAP web service using a static WSDL.

- [Create a scripted web service using a static WSDL](#)

To use a static WSDL, create a scripted web service.

- [Create a static WSDL](#)

Create a static WSDL with the required format to override the standard WSDL for your scripted web service.

- [Create a static WSDL script include](#)

Create a script include to define the majority of the code used to process static WSDL requests.

- [Use the static WSDL](#)

Load the static WSDL into a SOAP client to make requests to the SOAP web service.

To use a static WSDL, create a scripted web service.

Before you begin

Role required: web_service_admin or admin

Procedure

1. Navigate to **All > System Web Service > Scripted Web Services**.
 2. Click **New**.
 3. Enter a **Name** for the scripted SOAP web service such as `FakeStockValue`.
 4. Enter a **Script** for the web service to run.
 5. Click **Submit**.
- [Scripted web service example](#)

This example demonstrates the processing script for the `FakeStockValue` web service.

This example demonstrates the processing script for the `FakeStockValue` web service.

```
var vProcessor = new FakeStockValue (soapRequestXML) ;  
  
    var responseElement = vProcessor. process ( ) ; if (res  
ponseElement != null ) {  
        response. soapResponseElement = responseElement ; } els  
e {  
        response. soapResponseElement = vProcessor. generateSoap  
Fault ( "unknown error" ) ; }
```

Create a static WSDL with the required format to override the standard WSDL for your scripted web service.

Before you begin

Role required: web_service_admin or admin

Procedure

1. Navigate to **All > System Web Services > Static WSDL**.
 2. Create a static WSDL record using the same name as the scripted web service, such as `FakeStockValue`.
 3. Enter the custom WSDL into the **WSDL** field.
 4. Click **Submit**.
- [Static WSDL example](#)

This example demonstrates the `FakeStockValue` WSDL.

This example demonstrates the `FakeStockValue` WSDL.

```
<?xml version= "1.0" ?><definitions name = "StockQuote" targetNamespace = "http://example.com/stockquote.wsdl" xmlns:tns = "http://example.com/stockquote.wsdl" xmlns:xsd1 = "http://example.com/stockquote.xsd" xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/" xmlns = "http://schemas.xmlsoap.org/wsdl/" >

    <types><schema targetNamespace = "http://example.com/stockquote.xsd" xmlns = "http://www.w3.org/2000/10/XMLSchema" ><element name = "TradePriceRequest" ><complexType><all><element name = "tickerSymbol" type = "string" /></all></complexType></element><element name = "TradePrice" ><complexType><all><element name = "price" type = "float" /></all></complexType></element></schema></types>

    <message name = "GetLastTradePriceInput" ><part name = "body" element = "xsd1:TradePriceRequest" /></message>

    <message name = "GetLastTradePriceOutput" ><part name = "body" element = "xsd1:TradePrice" /></message>

    <portType name = "StockQuotePortType" ><operation name = "GetLastTradePrice" ><input message = "tns:GetLastTradeP
```

```
riceInput" /><output message = "tns:GetLastTradePriceOutput" /></operation></portType>

<binding name = "StockQuoteSoapBinding" type = "tns:StockQuotePortType"><soap:binding style = "document" transport = "http://schemas.xmlsoap.org/soap/http" /><operation name = "GetLastTradePrice" ><soap:operation soapAction = "" /><input><soap:body use = "literal" /></input><output><soap:body use = "literal" /></output></operation></binding>

<service name = "StockQuoteService" ><documentation>My first service</documentation><port name = "StockQuotePort" binding = "tns:StockQuoteSoapBinding" ><soap:address location = "https://myinstance.service-now.com/FakeStockValue.do?SOAP" /></port></service>

</definitions>
```

Create a script include to define the majority of the code used to process static WSDL requests.

Before you begin

Role required: script_include_admin or admin

About this task

By implementing the majority of the custom functionality in a script include, you can reuse the script include in multiple areas.

Procedure

1. Navigate to **All > System UI > Script Includes**.
 2. Click **New**.
 3. Enter a **Name** for the script include that matches the name of the static WSDL, such as `FakeStockValue`.
 4. Enter the script include code in the **Script** field.
 5. Click **Submit**.
- Static WSDL script include example

This example demonstrates the FakeStockValue script include that implements much of the static WSDL behavior.

This example demonstrates the FakeStockValue script include that implements much of the static WSDL behavior.

```
var FakeStockValue = Class.create();

FakeStockValue.prototype = {
    initialize : function(requestXML) {
        //Use some backend XML utilities...you could use string tools if you wish
        this.xmlutil = Packages.com.glide.util.XMLUtil;
        //converting the string to an XML Document
        this.fSoapDoc = new XMLDocument(requestXML);
    },

    process : function() {
        var soapBody = this.fSoapDoc.getNode("/Envelope/Body");
        //
        //Our WSDL was formatted to have the only first child element be the function
        var funcNode = this.xmlutil.getFirstChildElement(soapBody);
        var nodeName = this.xmlutil.getNodeNameNS(funcNode);

        //If the function for this SOAP request is TradePriceRequest, perform the necessary actions
        if (nodeName == "TradePriceRequest") {
            return this.fakeOutTradePriceRequest(funcNode);
        }

        //Couldn't find any supported functions in this SOAP request
        return this.generateSoapFault("un-supported API call:" + nodeName);
    },

    fakeOutTradePriceRequest : function (funcNode) {
        //Create the beginnings of our XML response
        var r = new XMLDocument("<GetLastTradePriceOutput xmlns='https://www.service-now.com/vws/FakeStockValue'/'>");
    }
}
```

```
//Do the necessary actions here...we're going to get the USER ID of the user
    //used to make this SOAP call. Then we will return the
e
//stock symbol they were asking about
var usersysid = gs.getUserID();
var now_GR = new GlideRecord("sys_user");
gr.get(usersysid);
var username = gr.user_name;
var quoteSymbol = this.xmlutil.getText(funcNode);
//Create a "message" element to store our response message
r.createElement("message", username + ", You were looking for a quote on "+quoteSymbol);
return r.getDocumentElement();
},

generateSoapFault : function (str) {
    var f = "<SOAP-ENV:Fault>" +
        "<faultcode xsi:type='xsd:string'>SOAP-ENV:FakeStockValue</faultcode>" +
        "<faultstring xsi:type='xsd:string'>" + str +
        "</faultstring>" +
        "</SOAP-ENV:Fault>"
    var s = new XMLDocument(f);
    return s.getDocumentElement();
}
}
```

initialize function

The initialize function takes the XML request string and converts it to an XML Document object that you can navigate and manipulate using libraries. Alternatively, you can leave the XML request as a string and navigate it using regular expressions.

process function

The process function is called by the scripted web service. This function grabs the first child element in the XML after the body element. The WSDL uses this child element to determine which function to use. In this WSDL there is only one possible function but most WSDLs provide

many functions. If more functions were available, there would be more "if" statements that tested the first child element for the various function names.

fakeOutTradePriceRequest function

The fakeOutTradePriceRequest function is the implementation of the only available function in the WSDL. This function looks up the user that the SOAP request authenticated as and retrieves the **user_name** then returns it to the SOAP client. The fakeOutTradePriceRequest function could be expanded to perform useful activities, such as looking up a stock symbol and returning the last traded price.

generateSoapFault function

The generateSoapFault function returns a SOAP error that can be called if there are problems.

Load the static WSDL into a SOAP client to make requests to the SOAP web service.

The web service client provides

- The FakeStockValue project.
- The StockQuoteBinding web service.
- The GetLastTradePrice SOAP function. This function generates request records when run.

Loaded WSDL



You can change the default request XML in the static WSDL to include a stock symbol.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:stoc="http://example.com/stockquote.xsd">
  <soapenv:Header/>
  <soapenv:Body>
    <stoc:TradePriceRequest>IBM</stoc:TradePriceRequest>
```

```
</soapenv:Body>
</soapenv:Envelope>
```

Submitting a SOAP request to this web service endpoint returns the following to the requesting SOAP client.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Body>
        <GetLastTradePriceOutput xmlns="https://www.service-now.com/vws/FakeStockValue">
            <message>admin2, You were looking for a quote on IBM</message>
        </GetLastTradePriceOutput>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Direct web services

A direct web service is available for any table in the system if the correct access control list is configured.

The supported format of the incoming message is document style literal XML SOAP documents (Document/Literal). To retrieve a direct web service WSDL description and XML schema, point to the relative URL <tablename>.do?WSDL. For example, to retrieve the WSDL for the Incident table on the online demo system, use the following URL: https://<instance name>.service-now.com/incident.do?WSDL.

Extended query parameters

Extended query parameters enable you to filter and modify the return results of a SOAP query when using the get, getKeys, and getRecords functions.

Note: Extended query element names are preceded by two underscore characters.

Available extended query parameters

Parameter	Description	Example
<code>__encoded_query</code>	Specify an encoded query string to be used in filtering the returned results. The encoded query string format is similar to the value that may be specified in a <code>sysparm_query</code> URL parameter. See the encoded query building example in the RSS feed generator examples.	<pre><__encoded_query>active=true^category='hardware'</__encoded_query></pre>
<code>__order_by</code>	Instruct the returned results to be ordered by the specified field.	<pre><__order_by>priority</__order_by></pre>
<code>__order_by_desc</code>	Instruct the returned results to be ordered by the specified field, in descending order.	<pre><__order_by_desc>opened_date</__order_by_desc></pre>
<code>__exclude_columns</code>	Specify a list of comma delimited field names to exclude from the result set.	<pre><__exclude_columns>sys_created_on,sys_created_by,caller_id,priority</__exclude_columns></pre>
<code>__limit</code>	Limit the number of records that are returned.	<pre><__limit>100</__limit></pre>
<code>__first_row</code>	Instruct the results to be offset by this number of records from the beginning of the set. When used with <code>__last_row</code> has	<pre><__first_row>250</__first_row></pre>

Parameter	Description	Example
	the effect of querying for a window of results. The results are inclusive of the first row number.	
<code>_last_row</code>	Instruct the results to be limited by this number of records from the beginning of the set, or the <code>_start_row</code> value when specified. When used with <code>_first_row</code> has the effect of querying for a window of results. The results are less than the last row number, and does not include the last row.	<code><_last_row>500</_last_row></code>
<code>_use_view</code>	Specify a Form view by name, to be used for limiting and expanding the results returned. When the form view contains deep referenced fields such as <code>caller_id.email</code> , this field will be returned in the result as well.	<code><_use_view>soap_view</_use_view></code>

Direct web services namespace

Specifying a unique namespace for each table

The `glide.wsdl.definition.use_unique_namespace` property ensures that each table's direct web service WSDL has a unique targetNamespace attribute. This property is **true** by default, which requires a table's direct web service WSDL to use a targetNamespace value of `http://`

www.service-now.com/<table name>. When **false** (or when the property is not present), all tables use the same targetNamespace value of <http://www.service-now.com>. Since all tables also share the same operation names, a web service client attempting to consume more than one ServiceNow web service would be unable to differentiate between requests between multiple tables. Using a unique targetNamespace value allows web service clients to distinguish requests between multiple tables.

For example, the direct web service WSDL for the incident table uses this targetNamespace value:

```
<wsdl:definitions xmlns:soapenc= "http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/" xmlns:http = "http://schemas.xmlsoap.org/wsdl/http/" xmlns:tns = "http://www.service-now.com/incident" xmlns:xs d = "http://www.w3.org/2001/XMLSchema" xmlns:mime = "http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/" targetNamespace = "http://www.service-now.com/incident" ><wsdl:types><xsd:schema elementFormDefault = "unqualified" targetNamespace = "http://www.service-now.com/incident" >
```

Setting namespace requirements

ServiceNow's WSDL schema by default declares an attribute of elementFormDefault="unqualified". This attribute indicates whether or not locally declared elements must be qualified by the targetNamespace in an instance document. If the value of this attribute is **unqualified**, then locally declared elements should not be qualified by the targetNamespace. If the value of this attribute is **qualified**, then locally declared elements must be qualified by the targetNamespace.

However, this is incompatible with the way clients generated from WSDL (for example, .NET, Axis2, webMethods) process the embedded schema. It removes the schema namespace as a result, making the web service response unparseable.

To overcome this compatibility issue, a boolean property called `glide.wsdl.schema.UnqualifiedElementFormDefault` is introduced. This property has the value of **true** by default. Setting it to **false** enables clients generated from WSDL to parse the return value of the web service invocation. You can modify this property using the Web Services properties page at **System Properties > Web Services**.

Allowing duplicate service names

By default, service names from dynamically generated WSDL are unique and have the following format:

```
ServiceNow_<table name>
```

To allow duplicate service names, administrators can set the `glide.wsdl.unique_service_name` property to **false**. Create the property if it does not exist.

- [Use forms to limit or extend the query response](#)

On occasion, there is a need to limit the number of field values that a SOAP query returns.

- [Return the display value for reference variables](#)

When you query a record using a get or getRecords function, the instance returns all fields associated with that record. The fields are often reference fields that contain a `sys_id` for a record on another table.

- [Clear values from a target instance](#)

You can pass an empty value through a SOAP parameter to clear the respective value in the target instance.

- [Retrieve journal entries using direct web services](#)

To get the contents of a journal field, make a second soap request against the `sys_journal_field` table to pull the appropriate journal records back for the record in question.

- [Retrieve choice fields using direct web services](#)

To retrieve or set choice fields, use the choice **Value**, not the **Label**.

- [Persist an HTTP session across all SOAP calls](#)

In circumstances when a SOAP client makes many calls in a short amount of time, you may want to re-use a single HTTP session for all SOAP calls.

- [SOAP direct web service API functions](#)

The standard SOAP API is a set of small, globally defined functions that can be performed on a targeted resource.

Related concepts

- [SOAP web service](#)

On occasion, there is a need to limit the number of field values that a SOAP query returns.

You can use a form view to limit the number of field values returned by a SOAP query. Specifying a form view has the effects of:

1. Limiting the response elements to contain only the fields on the view.
2. Specifying reference record field values from referenced fields such as **caller_id.email**. This causes the value of the caller's email to be returned in the SOAP response.

To enable form views for SOAP queries, configure the `com.glide.soap.view` property to contain the name of the view that you want to use for all SOAP query responses. The default is `soap_response`. You can also specify the view name as a URL parameter, `sysparm_view=<view name>`, when making the SOAP call. For example:

```
https://<instance name>.service-now.com/incident.do?SOAP&sysparm_view=ess
```

By default, if a specified view name does not exist, the response contains all fields.

Related concepts

- [Return the display value for reference variables](#)
- [Clear values from a target instance](#)
- [Retrieve journal entries using direct web services](#)
- [Retrieve choice fields using direct web services](#)

- Persist an HTTP session across all SOAP calls

Related reference

- [SOAP direct web service API functions](#)

When you query a record using a get or getRecords function, the instance returns all fields associated with that record. The fields are often reference fields that contain a sys_id for a record on another table.

Use one of these options if you want the display value for the field to be returned instead of the sys_id:

1. Add the glide.soap.return_displayValue property to your system properties, and every SOAP request will return a display value for a reference field.
2. Add the displayvalue=true parameter to your SOAP request URL, and SOAP requests with that parameter will return a display value for a reference field as a string, instead of the sys_id. The SOAP URL would look as follows: <https://<instance name>.service-now.com/incident.do?displayvalue=true&SOAP>.
3. Add the displayvalue=all parameter to your SOAP request URL, and SOAP requests with that parameter will return a display value for a reference field, in addition to the sys_id. The response element name for the display value field will be prefixed with dv, such as dv_caller_id.

Related concepts

- [Use forms to limit or extend the query response](#)
- [Clear values from a target instance](#)
- [Retrieve journal entries using direct web services](#)
- [Retrieve choice fields using direct web services](#)
- Persist an HTTP session across all SOAP calls

Related reference

- [SOAP direct web service API functions](#)

You can pass an empty value through a SOAP parameter to clear the respective value in the target instance.

You can also pass an empty (null) value through the &allow_empty_value=true SOAP query parameter to clear the respective value in the target instance.

For example, https://<instance name>.service-now.com/incident.do?SOAP&allow_empty_value=true lets you pass an empty value to the incident record in an instance.

You can then enter lines like the following in the SOAP request:

```
<assigned_to>value</assigned_to>
<assignment_group>value</assignment_group>
<category></category>
```

In the above example,

- <assigned_to>value</assigned_to> changes the value in the **Assigned to** field to the value specified in the SOAP request.
- <assignment_group>value</assignment_group> changes the value in the **Assignment group** field to the value specified in the SOAP request.
- <category></category> clears the value in the **Category** field.

Related concepts

- [Use forms to limit or extend the query response](#)
- [Return the display value for reference variables](#)
- [Retrieve journal entries using direct web services](#)
- [Retrieve choice fields using direct web services](#)
- [Persist an HTTP session across all SOAP calls](#)

Related reference

- [SOAP direct web service API functions](#)

To get the contents of a journal field, make a second soap request against the sys_journal_field table to pull the appropriate journal records back for the record in question.

The URL for the WSDL would be in the following format.

```
https://instance-name.service-now.com/sys_journal_field.do  
?WSDL
```

To retrieve the journal entries, you will first need to query the incident for its sys_id value, and then supply it as the element_id value in a getRecords call. To specify records only for the **comments** field, specify the value comments for the **element** field. For example, a SOAP request would look like the following:

```
<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/" xmlns:sys= "http://www.service-now.com/sys_journal_field" ><soapenv:Header /><soapenv:Body><sys:getRecords><element>comments</element><element_id>9d385017c611228701d22104cc95c371</element_id></sys:getRecords></soapenv:Body></soapenv:Envelope>
```

Related concepts

- [Use forms to limit or extend the query response](#)
- [Return the display value for reference variables](#)
- [Clear values from a target instance](#)
- [Retrieve choice fields using direct web services](#)
- [Persist an HTTP session across all SOAP calls](#)

Related reference

- SOAP direct web service API functions

To retrieve or set choice fields, use the choice **Value**, not the **Label**.

For example, if you want to retrieve a list of all closed incidents, use the numerical value for **Closed**, which is **7** by default.

```
<state>7</state>
```

To see a list of choice values:

1. Navigate to the form containing the choice field. For example, navigate to **Incident > Open** and select an incident.
2. Right-click the choice value field and select **Configure Dictionary**. For example, configure the dictionary for the **State** field.
3. From the Choices related list, note the value for the label you want to query. For example, note that the **Closed** choice has a value of **7**.

Choices	Go to	Table	Element	Language	Value	Label
▶ Choices						
<input type="checkbox"/>		incident	state	en	1	New
<input type="checkbox"/>		incident	state	en	2	Active
<input type="checkbox"/>		incident	state	en	3	Awaiting Problem
<input type="checkbox"/>		incident	state	en	4	Awaiting User Info
<input type="checkbox"/>		incident	state	en	5	Awaiting Evidence
<input type="checkbox"/>		incident	state	en	6	Resolved
<input type="checkbox"/>		incident	state	en	7	Closed
<input type="checkbox"/>		problem	state	en	1	Open
<input type="checkbox"/>		problem	state	en	3	Pending Change
<input type="checkbox"/>		problem	state	en	2	Known Error
<input type="checkbox"/>		problem	state	en	4	Closed/Resolved
<input type="checkbox"/>		task	state	en	-5	Pending
<input type="checkbox"/>		task	state	en	1	Open
<input type="checkbox"/>		task	state	en	2	Work in Progress
<input type="checkbox"/>		task	state	en	3	Closed Complete
<input type="checkbox"/>		task	state	en	4	Closed Incomplete
<input type="checkbox"/>		task	state	en	7	Closed Skipped

Related concepts

- Use forms to limit or extend the query response
- Return the display value for reference variables
- Clear values from a target instance
- Retrieve journal entries using direct web services
- Persist an HTTP session across all SOAP calls

Related reference

- [SOAP direct web service API functions](#)

In circumstances when a SOAP client makes many calls in a short amount of time, you may want to re-use a single HTTP session for all SOAP calls.

Each SOAP call creates a new user session that persists until it expires. To create a single user session and re-use it for all inbound SOAP calls, develop your SOAP client following these guidelines:

- Use a module like `HTTP::Cookies` to create a "cookie jar."
- Save the cookies returned by ServiceNow after each request (handled automatically by `HTTP::Cookies`).
- Re-send the cookies in the cookie jar with each subsequent request.

Note: If you have enabled the session rotation high security setting, it will immediately invalidate the `JSESSIONID` returned from the server with the first response header. The second response includes a new `JSESSIONID`.

In perl, you can automatically save and send cookies with the following code:

```
use HTTP::Cookies;

#we want to store and re-send cookies
my $cookies = HTTP::Cookies->new(ignore_discard => 1);

my $soap = SOAP::Lite
```

```
-> proxy('http://localhost:8080/glide/ecc_queue.do?SOAP');  
  
#Set the cookie jar  
$soap->transport->cookie_jar($cookies);
```

Related concepts

- Use forms to limit or extend the query response
- Return the display value for reference variables
- Clear values from a target instance
- Retrieve journal entries using direct web services
- Retrieve choice fields using direct web services

Related reference

- [SOAP direct web service API functions](#)

The standard SOAP API is a set of small, globally defined functions that can be performed on a targeted resource.

The targeted resource (or table) is defined in the URL by the format <https://<instance name>.service-now.com/<table name>.do>.

Data Modification API

Method Summary	Description
insert	Creates a new record for the table targeted in the URL.
insertMultiple	Creates multiple new records for the table targeted in the URL. To enable multiple inserts, activate the Web service import sets .
update	Updates a existing record in the targeted table in the URL.

Method Summary	Description
	identified by the mandatory sys_id field.
deleteRecord	Deletes a record from the targeted table by supplying its <code>sys_id</code> .
deleteMultiple	Delete multiple records from the targeted table by example values.

Data Retrieval API

Method Summary	Description
getKeys	Query the targeted table by example values and return a comma delimited <code>sys_id</code> list.
getRecords	Query the targeted table by example values and return all matching records and their fields.
get	Query a single record from the targeted table by <code>sys_id</code> and return the record and its fields.
aggregate	Query using aggregate functions SUM, COUNT MIN, MAX, LAST, and AVG. To enable the aggregate functions, activate the Aggregate Web Service Plugin.

- [Data Retrieval API](#)

Data Retrieval API method summaries and descriptions.

- [Data Modification API](#)

Data Modification API method summaries and descriptions.

Related concepts

- Use forms to limit or extend the query response
- Return the display value for reference variables
- Clear values from a target instance
- Retrieve journal entries using direct web services
- Retrieve choice fields using direct web services
- Persist an HTTP session across all SOAP calls

Data Retrieval API method summaries and descriptions.

Data Retrieval API

Method Summary	Description
getKeys	Query the targeted table by example values and return a comma delimited sys_id list.
getRecords	Query the targeted table by example values and return all matching records and their fields.
get	Query a single record from the targeted table by sys_id and return the record and its fields.
aggregate	Query using aggregate functions SUM, COUNT MIN, MAX, LAST, and AVG. To enable the aggregate functions, activate the Aggregate Web Service Plugin.

- [getKeys](#)

Query the targeted table by example values and return a comma delimited sys_id list.

- [getRecords](#)

Query the targeted table by example values and return all matching records and their fields.

- [get](#)

Query a single record from the targeted table by `sys_id` and return the record and its fields.

- [aggregate](#)

Query a table using an aggregate function including SUM, COUNT, MIN, MAX, LAST, and AVG.

Related reference

- [Data Modification API](#)

Query the targeted table by example values and return a comma delimited `sys_id` list.

Input fields

Any field value in the targeted table.

Output fields

A SOAP response element `sys_id` that contains a comma delimited list of [Unique record identifier \(`sys_id`\)](#) values.

Sample SOAP messages

Example

Sample SOAP request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
    <soapenv:Header/>
    <soapenv:Body>
        <inc:getKeys>
```

```
<category>hardware</category>
</inc:getKeys>
</soapenv:Body>
</soapenv:Envelope>
```

Example

Sample SOAP response

```
<soapenv:Envelope xmlns:inc="http://www.service-now.com/incident" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <getKeysResponse>
      <sys_id>46e18c0fa9fe19810066a0083f76bd56,46e57642a9fe1981000b96a5dca501ff,46f1784ba9fe19810018aa27fbb23482</sys_id>
      <count>7</count>
    </getKeysResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Language-specific sample messages

For language-specific getKeys samples, refer to the following topics:

[Perl SOAP::Lite](#)

[Java Apache Axis2](#)

[Python](#)

Query the targeted table by example values and return all matching records and their fields.

Input fields

Any field value in the targeted table.

Output fields

The getRecordResponse element may contain one or more getRecordsResult elements that encapsulate elements representing the field values of records matching the query.

Sample SOAP messages

Example

Sample SOAP request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
    <soapenv:Header/>
    <soapenv:Body>
        <inc:getRecords>
            <number>INC0000002</number>
        </inc:getRecords>
    </soapenv:Body>
</soapenv:Envelope>
```

Example

Sample SOAP request using an encoded query to filter where incident number is INC0000001 or INC0000002

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
    <soapenv:Header/>
    <soapenv:Body>
        <inc:getRecords>
            <__encoded_query>number=INC0000001^ORnumber=INC000002</__encoded_query>
        </inc:getRecords>
    </soapenv:Body>
</soapenv:Envelope>
```

Example

Sample SOAP response that contains 1 record

```
<soapenv:Envelope xmlns:inc="http://www.service-now.com/incident" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <getRecordsResponse>
      <getRecordsResult>
        <caller_id>5137153cc611227c000bbd1bd8cd2007</caller_id>
        <caller_id.email>david.loo@service-now.com</caller_id.email>
        <closed_at/>
        <number>INC0000002</number>
        <opened_at>2009-12-14 23:07:12</opened_at>
        <short_description>Can't get to network file shares</short_description>
      </getRecordsResult>
    </getRecordsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Example

Sample SOAP response that contains more than 1 record

```
<soapenv:Envelope xmlns:inc="http://www.service-now.com/incident" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <getRecordsResponse>
      <getRecordsResult>
        <caller_id>5137153cc611227c000bbd1bd8cd2006</caller_id>
        <caller_id.email>rick.berzle@yourcompany.com</caller_id.email>
        <closed_at>2009-12-17 22:55:16</closed_at>
        <number>INC0000009</number>
        <opened_at>2009-12-16 22:50:23</opened_at>
        <short_description>Reset my password</short_description>
      </getRecordsResult>
    <getRecordsResult>
```

```
        <caller_id>5137153cc611227c000bbd1bd8cd2005</c
aller_id>
        <caller_id.email>fred.luddy@yourcompany.com</c
aller_id.email>
        <closed_at>2009-12-15 22:54:55</closed_at>
        <number>INC0000010</number>
        <opened_at>2009-12-10 22:53:02</opened_at>
        <short_description>Need Oracle 10GR2 installed
</short_description>
    </getRecordsResult>
</getRecordsResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Language-specific sample messages

For language-specific getRecords samples, refer to the following topics:

[Perl SOAP::Lite](#)

[Java Apache Axis2](#)

[Microsoft .NET web services client examples](#)

[Python](#)

Query a single record from the targeted table by `sys_id` and return the record and its fields.

Input fields

An element `<sys_id>` identifying the `sys_id` of the record to be retrieved.

Output fields

A `getResponse` element encapsulating all field values for the record retrieved.

Sample SOAP messages

Example

Sample SOAP request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
  <soapenv:Header/>
  <soapenv:Body>
    <inc:get>
      <sys_id>46e18c0fa9fe19810066a0083f76bd56</sys_id>
    </inc:get>
  </soapenv:Body>
</soapenv:Envelope>
```

Example

The resulting response of a get function call looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <getResponse xmlns="http://www.service-now.com/incident">
      <active>1</active>
      <approval>not requested</approval>
      <assigned_to>46c6f9efa9fe198101ddf5eed9adf6e7</assigned_to>
      <caller_id>46b673a6a9fe19810007ab03cbd5849d</caller_id>
      <category>network</category>
      <cmdb_ci>0c43f35dc61122750182c132a29e3243</cmdb_ci>
      <comments>Testing</comments>
      <contact_type>phone</contact_type>
      <due_date>2007-10-28 13:29:45</due_date>
      <escalation>0</escalation>
      <impact>3</impact>
      <incident_state>1</incident_state>
      <knowledge>0</knowledge>
      <location>1081761cc611227501d063fd3475a2de</location>
      <made_sla>1</made_sla>
      <notify>1</notify>
    </getResponse>
  </soap:Body>
</soap:Envelope>
```

```
<number>INC10055</number>
<opened_at>2007-09-18 00:32:09</opened_at>
<opened_by>46bac3d6a9fe1981005f299d979b8869</opened_
by>
<priority>0</priority>
<reassignment_count>0</reassignment_count>
<severity>0</severity>
</getResponse>
</soap:Body>
</soap:Envelope>
```

Query a table using an aggregate function including SUM, COUNT, MIN, MAX, LAST, and AVG.

Note: Functionality described here requires the Aggregate Web Service plugin.

Input fields

Any element of the target table. In addition, one or more of the aggregate functions (SUM, COUNT, MIN, MAX, LAST, and AVG).

A GROUP BY and a HAVING clause may also be added.

Output fields

An aggregateResponse element encapsulating all field values for the record retrieved.

Sample SOAP messages

Example

Sample SOAP request using COUNT aggregate function.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap="http://schemas.xmlsoap.org/
soap/encoding/"
                     xmlns:SOAP-ENV="http://schemas.xmlsoap.
org/soap/envelope/"
                     xmlns:m="http://www.service-now.com"
                     xmlns:tns="http://www.service-now.com/m
ap"
```

```
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
hema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
hema-instance"
            SOAP-ENV:encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<aggregate>
<COUNT>number</COUNT>
<active>true</active>
</aggregate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example

The resulting response of a COUNT aggregate function call looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.
org/soap/envelope/"
                    xmlns:SOAP-ENC="http://schemas.xmlsoap.
org/soap/encoding/"
                    xmlns:m="http://www.service-now.com"
                    xmlns:tns="http://www.service-now.com/m
ap"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
hema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
hema-instance"
            SOAP-ENV:encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<aggregateResponse>
<aggregateResult>
<avg>2.7200</avg>
</aggregateResult>
</aggregateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example

Sample SOAP request using AVG aggregate function with a GROUP BY clause.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                     xmlns:m="http://www.service-now.com"
                     xmlns:tns="http://www.service-now.com/m
ap"
                     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema
schema-instance"
                     SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <aggregate xmlns="http://www.service-now.com">
            <GROUP_BY>category</GROUP_BY>
            <active>true</active>
            <AVG>severity</AVG>
        </aggregate>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example

The resulting response of a AVG aggregate function call with a GROUP BY clause looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                     xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                     xmlns:m="http://www.service-now.com"
                     xmlns:tns="http://www.service-now.com/m
ap"
                     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
    "hema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <SOAP-ENV:Body>
                <aggregateResponse>
                    <aggregateResult>
                        <avg>1.0000</avg>
                        <category>database</category>
                    </aggregateResult>
                    <aggregateResult>
                        <avg>3.0000</avg>
                        <category>hardware</category>
                    </aggregateResult>
                    <aggregateResult>
                        <avg>3.0000</avg>
                        <category>inquiry</category>
                    </aggregateResult>
                    <aggregateResult>
                        <avg>2.0000</avg>
                        <category>network</category>
                    </aggregateResult>
                    <aggregateResult>
                        <avg>2.6923</avg>
                        <category>software</category>
                    </aggregateResult>
                </aggregateResponse>
            </SOAP-ENV:Body>
        </SOAP-ENV:Envelope>
```

Example

Sample SOAP request using an encoded query to filter the aggregate:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap="http://schemas.xmlsoap.org/
soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:m="http://www.service-now.com"
xmlns:tns="http://www.service-now.com/m
ap">
```

```
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
hema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
hema-instance"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<aggregate>
<COUNT>number</COUNT>
<active>true</active>
<__encoded_query>number=INC0000001^ORnumber=INC0000002</__encoded_query>
</aggregate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example

Sample aggregate request using HAVING to narrow the results.

HAVING takes four fields. Each field is delimited by "^": the aggregate type, the field of the aggregate, the operation type, and the value to compare.

More than one HAVING can be added to the request, so you can use HAVING expressions, but there is no support for OR.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/encoding/">
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
        xmlns:m="http://www.service-now.com"
        xmlns:tns="http://www.service-now.com/mpl"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
hema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
hema-instance"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<aggregate>
<COUNT>sys_id</COUNT>
```

```
<GROUP_BY>internal_type</GROUP_BY>
<HAVING>COUNT^*^>^10</HAVING>
<HAVING>COUNT^*^<^20</HAVING>
<COUNT>sys_id</COUNT>
<active>true</active>
</aggregate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Data Modification API method summaries and descriptions.

Data Modification API

Method Summary	Description
insert	Creates a new record for the table targeted in the URL.
insertMultiple	Creates multiple new records for the table targeted in the URL. To enable multiple inserts, activate the Web service import sets .
update	Updates a existing record in the targeted table in the URL, identified by the mandatory sys_id field.
deleteRecord	Deletes a record from the targeted table by supplying its sys_id .
deleteMultiple	Delete multiple records from the targeted table by example values.

- [insert](#)

Creates a new record for the table targeted in the URL.

- [insertMultiple](#)

Creates multiple new records for the table targeted in the URL.

- [update](#)

Updates an existing record in the targeted table in the URL, identified by the mandatory **sys_id** field.

- [deleteRecord](#)

Delete a record from the targeted table by supplying its **sys_id**.

- [deleteMultiple](#)

Delete multiple records from the targeted table by example values.

Related reference

- [Data Retrieval API](#)

Creates a new record for the table targeted in the URL.

Input fields

All fields from the targeted table, excluding system fields. Fields configured as mandatory in the System Dictionary are reflected in the WSDL with the attribute `minOccurs=1`.

Output fields

Insert method output fields

Table type	Output fields
Regular	The sys_id field and the display value of the target table (<code>table</code>) are returned.
Import set	The sys_id of the import set row, the name of the transformed target table (<code>table</code>), the display_name for the transformed target table, the display_value of the transformed target row, and a status field, which can contain inserted , updated , or error .

Table type	Output fields
	<p>There can be an optional status_message field or an error_message field value when status=error.</p> <p>When an insert did not cause a target row to be transformed (skipped because a key value is not specified), the sys_id field will contain the sys_id of the import set row, rather than the targeted transform table.</p>
Import set with multiple transforms	The response from this type of insert will contain multiple sets of fields from the regular import set table insert wrapped in a multiInsertResponse parent element. Each set will contain a map field, showing which transform map created the response.

Sample SOAP messages for a regular table

Example

The following example shows an insert that specifies the short description only:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope xmlns:tns="http://www.service-now.com/i
ncident"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding"
  xmlns:m="http://www.service-now.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xm
lns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/en
coding/">
<SOAP-ENV:Body>
<insert xmlns="http://www.service-now.com">
    <short_description xsi:type="xsd:string">This
is a test</short_description>
</insert>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example

The resulting response looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding
/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope
/
    xmlns:m="http://www.service-now.com"
    xmlns:tns="http://www.service-now.com/incident" xmlns:xs
d="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Body>
        <insertResponse xmlns="http://www.service-now.com">
            <sys_id>6b06494fc611227d00b5f87caf618831</sys_
id>
        </insertResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Language-specific sample messages

For language-specific insert samples, refer to the following topics:

[Perl SOAP::Lite](#)

[Java Apache Axis2](#)

[Python](#)

Creates multiple new records for the table targeted in the URL.

Input fields

The `insertMultiple` element may contain 1 or more record tags that contains all fields from the targeted table, excluding system fields. Limit the number of records inserted in a single operation to no more than 200. You can gradually increase this number with subsequent exports if the increase does not negatively impact instance performance.

Output fields

The `insertMultipleResponse` tag is followed by 1 or more record tags that contains:

Insert method output fields

Table type	Output fields
Regular	The sys_id field and the display value of the target table (<code>table</code>) are returned.
Import set	<p>The sys_id of the import set row, the name of the transformed target table (<code>table</code>), the display_name for the transformed target table, the display_value of the transformed target row, and a status field, which can contain inserted, updated, or error.</p> <p>There can be an optional status_message field or an error_message field value when <code>status=error</code>.</p> <p>When an insert did not cause a target row to be transformed (skipped because a key value is not specified), the sys_id field will</p>

Table type	Output fields
	contain the sys_id of the import set row, rather than the targeted transform table.
Import set with multiple transforms	The response from this type of insert will contain multiple sets of fields from the regular import set table insert wrapped in a multiInsertResponse parent element. Each set will contain a map field, showing which transform map created the response.

Sample SOAP messages for a regular table

Example

The following example shows an insert that specifies the short description only:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
  <soapenv:Header/>
  <soapenv:Body>
    <inc:insertMultiple>
      <record>
        <short_description>this is test 1</short_description>
      </record>
      <record>
        <short_description>this is test 2</short_description>
      </record>
      <record>
        <short_description>this is test 3</short_description>
      </record>
    </inc:insertMultiple>
```

```
</soapenv:Body>  
</soapenv:Envelope>
```

Example

The resulting response looks like this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">  
    <soapenv:Header/>  
    <soapenv:Body>  
        <insertMultipleResponse>  
            <insertResponse>  
                <sys_id>168160ad4a36231200a89091281dc803</sys_id>  
                <number>INC0055180</number>  
            </insertResponse>  
            <insertResponse>  
                <sys_id>1681622e4a36231200a8909115e5c388</sys_id>  
                <number>INC0055181</number>  
            </insertResponse>  
            <insertResponse>  
                <sys_id>1681626e4a36231200a89091fa3c0aa8</sys_id>  
                <number>INC0055182</number>  
            </insertResponse>  
        </insertMultipleResponse>  
    </soapenv:Body>  
</soapenv:Envelope>
```

Sample SOAP messages for an import set table

Example

The following example shows an insert that specifies the short description only:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:imp="http://www.service-now.com/import_notification">  
    <soapenv:Header/>  
    <soapenv:Body>
```

```
<imp:insertMultiple>:-->
  <imp:record>
    <imp:message>one</imp:message>
    <imp:uuid>a</imp:uuid>
  </imp:record>
  <imp:record>
    <imp:message>two</imp:message>
    <imp:uuid>b</imp:uuid>
  </imp:record>
  <imp:record>
    <imp:message>three</imp:message>
    <imp:uuid>c</imp:uuid>
  </imp:record>
</imp:insertMultiple>
</soapenv:Body>
</soapenv:Envelope>
```

Example

The resulting response looks like this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:imp="http://www.service-now.com/imp_notification">
  <soapenv:Header/>
  <soapenv:Body>
    <insertMultipleResponse>
      <insertResponse>
        <sys_id>1296b3ab0a0a0b5b73e966fbfab7acde</sys_id>
        <table>incident</table>
        <display_name>number</display_name>
        <display_value>INC0010033</display_value>
        <status>ignored</status>
        <status_message>No field values changed</status_message>
      </insertResponse>
      <insertResponse>
        <sys_id>1296b48e0a0a0b5b62513bb5974a7d96</sys_id>
        <table>incident</table>
        <display_name>number</display_name>
        <display_value>INC0010034</display_value>
```

```
        <status>ignored</status>
        <status_message>No field values changed</status_
s_message>
    </insertResponse>
    <insertResponse>
        <sys_id>1296b58b0a0a0b5b468f534659538b9a</sys_
id>
        <table>incident</table>
        <display_name>number</display_name>
        <display_value>INC0010035</display_value>
        <status>ignored</status>
        <status_message>No field values changed</status_
s_message>
    </insertResponse>
</insertMultipleResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Updates an existing record in the targeted table in the URL, identified by the mandatory **sys_id** field.

Input fields

All fields from the targeted table, excluding system fields, which will be used for updating the existing record. The **sys_id** field is used to locate the existing record.

Output fields

Returns the **sys_id** of the record that was updated.

Sample SOAP messages

Example

Sample SOAP request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.or
g/soap/envelope/" xmlns:inc="http://www.service-now.com/in
cident">
    <soapenv:Header/>
    <soapenv:Body>
        <inc:update>
            <sys_id>46e18c0fa9fe19810066a0083f76bd56</sys_id>
```

```
<short_description>this is updated</short_descrip  
tion>  
</inc:update>  
</soapenv:Body>  
</soapenv:Envelope>
```

Example

Sample SOAP response

```
<soapenv:Envelope xmlns:inc="http://www.service-now.com/in  
cident" xmlns:soapenv="http://schemas.xmlsoap.org/soap/env  
elope/">  
  <soapenv:Header/>  
  <soapenv:Body>  
    <updateResponse>  
      <sys_id>46e18c0fa9fe19810066a0083f76bd56</sys_id>  
    </updateResponse>  
  </soapenv:Body>  
</soapenv:Envelope>
```

Language-specific sample messages

For language-specific update samples, refer to the following topics:

[Perl SOAP::Lite](#)

[Java Apache Axis2](#)

[Microsoft .NET](#)

[Python](#)

Delete a record from the targeted table by supplying its **sys_id**.

Input fields

An element <sys_id> identifying the **sys_id** of the record to be retrieved.

Output fields

A <count> element within the deleteRecordResponse parent element indicating the number of records deleted, this will always equal to "1" for deleteRecord.

Sample SOAP messages

Example

Sample SOAP request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
  <soapenv:Header/>
  <soapenv:Body>
    <inc:deleteRecord>
      <sys_id>46e18c0fa9fe19810066a0083f76bd56</sys_id>
    </inc:deleteRecord>
  </soapenv:Body>
</soapenv:Envelope>
```

Example

Sample SOAP response

```
<soapenv:Envelope xmlns:inc="http://www.service-now.com/incident" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <deleteRecordResponse>
      <count>1</count>
    </deleteRecordResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Language-specific sample messages

For language-specific deleteRecord samples, refer to the following topics:

[Perl SOAP::Lite](#)

[Java Apache Axis2](#)

[Microsoft .NET](#)

[Python](#)

Delete multiple records from the targeted table by example values.

Input fields

All fields from the targeted table, including system fields, are used in query-by-example (QBE) fashion to locate records to be deleted. Query example fields can have special prefixes to constrain the search function.

Output fields

A <count> element within the deleteRecordResponse parent element indicating the number of records deleted.

Sample SOAP messages

Example

Sample SOAP request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
    <soapenv:Header/>
    <soapenv:Body>
        <inc:deleteMultiple>
            <category>hardware</category>
        </inc:deleteMultiple>
    </soapenv:Body>
</soapenv:Envelope>
```

Example

Sample SOAP response

```
<soapenv:Envelope xmlns:inc="http://www.service-now.com/incident" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header/>
    <soapenv:Body>
        <deleteMultipleResponse>
            <count>6</count>
        </deleteMultipleResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

```
</soapenv:Body>  
</soapenv:Envelope>
```

Language-specific sample messages

For language-specific deleteRecord samples, refer to the following topics:

[Perl SOAP::Lite](#)

[Java Apache Axis2](#)

[Microsoft .NET](#)

[Python](#)

SOAP web service import sets

Web service import sets complement direct web services and scripted SOAP web services by providing a web service interface to import sets tables.

By default, this type of web service synchronously transforms the incoming data based on the associated transform maps. If the associated import set mode is set to Asynchronous, the behavior is to save the data for transformation at a later time. Web service import sets tables publish all default web service functions in the WSDL.

You can access a web service import set WSDL by specifying the import set table name + ".do?WSDL" on the URL.

For example:

```
http://<instance name>.service-now.com/imp_notification.do?  
WSDL.
```

AttachmentCreator SOAP web service

Attach documents to records in ServiceNow by sending a SOAP message targeting the ecc_queue table.

Important: The AttachmentCreator SOAP web service is not recommended. Instead, use the REST [Attachment API](#) to manage attachments with web services.

Using the AttachmentCreator SOAP web service, you can attach a single document to a message that is a maximum of 5 MB. The following is an example of a URL or target end point: `https://instance_name.service-now.com/ecc_queue.do?WSDL`

ecc_queue Fields for Attachment Creation

Field Name	Description	Value
agent	The name of the agent sending in the request, this can be any value since it is not used for processing.	AttachmentCreator
topic	The topic of the queue record, this value must be set to "AttachmentCreator" to trigger the attachment creation	AttachmentCreator
name	This field must contain a ":" delimited value of the file name, and its content-type	file_name.xls:application/vnd.ms-excel
source	This field must contain a ":" delimited value of the target table and its sys_id	incident:dd90c5d70a0a0b3900aac5aee704ae8
payload	This field must contain the Base 64 encoded string representing the object to be attached	the base 64 encoded string

Sending in the values described in the table above will attach an Excel file to the incident table for the record located by the sys_id dd90c5d70a0a0b39000aac5aee704ae8

Security

Like all other HTTP based web services available on the platform, the AttachmentCreator SOAP web service is required to authenticate using basic authentication by default. The user ID that is used for authentication will be subjected to access control in the same way as an interactive user.

To create attachments, the SOAP user must have any roles required to create Attachment [sys_attachment] records, as well as the soap_create role, and any roles required to read and write records on the target table, such as the itil role to add attachments to incident records. By default there is no single role allowing you to add attachments. You can create a role to explicitly allow adding attachments, then assign this role to the SOAP user.

File type security

You can control what file types users can attach by setting the glide.attachment.extensions and glide.security.file.mime_type.validation properties.

For these properties to apply to the AttachmentCreator web service, the property glide.attachment.enforce_security_validation must be set to true. This property is true by default.

Example SOAP Message

The following is an example of a SOAP message that would take a text file "john1.txt" of mime-type: text/plain and attach it to an Incident with a GUID of: e6eed6950a0a3c59006f32c8e3ff3cf9. Note the payload is the base64 encoding of the file itself.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ecc="http://www.service-now.com/ec_c_queue">
    <soapenv:Header />
    <soapenv:Body>
        <ecc:insert>
```

```
<agent>AttachmentCreator</agent>
<topic>AttachmentCreator</topic>
<name>john1.txt:text/plain</name>
<source>incident:e6eed6950a0a3c59006f32c8e3ff3
cf9</source>
<payload>SSB3b25kZXIgaWYgc2h1IGtub3ducyB3aGF0I
HNoZSdzIGRvaW5nIG5vdy4K</payload>
</ecc:insert>
</soapenv:Body>
</soapenv:Envelope>
```

Example Node.js Script

The following example Node.js script adds an attachment to an incident record. Run this script from a client computer, not an instance.

```
/*
 *
 * Node.js to ServiceNow attachment upload via SOAP
 *
 * Andrew Venables andrew.venables@servicenow.com
 * July 2014
 *
 * Version 1.0
 *
 */

var soap = require('soap'), // https://github.com/vpulim/node-soap
    mime = require('mime'), // https://github.com/broofa/node-mime
    fs = require("fs");

var WSDL_FILENAME = 'ecc_queue.xml'; // Goto https://instance-name.service-now.com/ecc_queue.do?WSDL and save a copy of the WSDL locally for simplicity
var DIRECTORY_CONTAINING_FILES = '/Users/andrew.venables/Documents/Uploads'; // Local path to the directory containing all the files we want to upload
var USERNAME = 'andy.venables'; // An admin user account on the instance
var PASSWORD = 'MY_PASSWORD'; // Password for above account
var TARGET_TABLE = 'incident'; // Target table to attach t
```

```
he files to
var TARGET_SYS_ID = '9d385017c611228701d22104cc95c371'; // Target record / sys_id to attach the files to. OOTB INC000002

var files_to_upload; // Global that will contain our list of files to be uploaded
var pos = 0; // Global pointer for our position in the files_to_upload list

// Create a SOAP client to use to post to the instance
soap.createClient(WSDL_FILENAME, function(err, client) { // Node uses callbacks
    if (err) console.error(err);

    // Set the username and password
    client.setSecurity(new soap.BasicAuthSecurity(USER_NAME, PASSWORD));

    // Read all the files in our source directory, will include . and ..
    files_to_upload = fs.readdirSync(DIRECTORY_CONTAINING_FILES);

    console.log('Files to upload: ' + files_to_upload.length + '\n');

    // Start iterating through the list of files to upload
    next(client);

});

// Process the next file in the files_to_upload array
// This is a neat way of dealing with Node and its expectation of callbacks
function next(client) {

    // Check we haven't reached the end
    if (pos >= files_to_upload.length) return;

    // Get the next file to upload
    var file_name = files_to_upload[pos];
```

```
// Increment the pointer to the next file
pos++;

console.log(pos + '/' + files_to_upload.length+ ' - 
Uploading file: ' + file_name);

// A blank file is the end of the list
if (file_name == '') return;

// Skip to the next file as this one is invalid
if (file_name == '.' || file_name == '..' || file_
name.indexOf('.') == 0)
    next(client);

// Get the file type using an module called mime
var file_type = mime.lookup(file_name);
console.log('  of type: ' + file_type);

var file_payload;
// Load the file into a buffer
fs.readFile(DIRECTORY_CONTAINING_FILES + '/' + fil
e_name, function(err, the_data) {
    if (err) console.error(err);

    // Encode the buffer to base64
    file_payload = new Buffer(the_data, 'binar
y').toString('base64');

    // Set the parameters before we call the W
    eb Service
    var parameters = {
        'agent':           'AttachmentCreator
',
        'topic':            'AttachmentCreator',
        'name':             file_name+'.'+file_type,
        'source':           TARGET_TABLE+'.'+TARGET_ SY
S_ID,
        'payload':          file_payload
    };

    console.log('      sending...')
    // Make the Web Service call, remember nod
e likes callbacks
    client.insert(parameters, function(err, re
```

```
sult) {
    if (err) console.error(err);

    console.log(result);

    // This file is done, next!
    next(client);
  );
}
}
```

Example Perl Script

The following example Perl script will create an attachment to an incident record.

```
use MIME::Base64;
use strict;
use SOAP::Lite;

# the ServiceNow instance
my $SNC_HOST = "https://instance_name.service-now.com";
my $base64;
my $buf;

# upload and attach a file on the local disk, base 64 encode it into a string first
open(FILE, "/Users/davidloo/Desktop/test_files/number_test.xls") or die "$!";
binmode FILE; #preserves file formatting on Windows
while (read(FILE, $buf, 60*57)) {
    $base64 .= encode_base64($buf);
}

# call the sub routine to attach
attach_incident();

sub attach_incident {
    # target the ecc_queue table
    my $soap = SOAP::Lite->proxy("$SNC_HOST/ecc_queue.do?SOAP");
    $soap->{_transport}->{_proxy}->{ssl_opts}->{verify_hostname} = 0;
    my $method = SOAP::Data->name('insert')->attr({xmlns =>
```

```
'http://www.service-now.com/' });

    # set the ecc_queue parameters
    my @params = (SOAP::Data->name(agent => 'AttachmentCreator'));
    push(@params, SOAP::Data->name(topic => 'AttachmentCreator') );

    # identify the file name and its mime type
    push(@params, SOAP::Data->name(name => 'number_test.xls:application/vnd.ms-excel') );

    # attach to the incident, specifying its sys_id
    push(@params, SOAP::Data->name(source => 'incident:dd90c5d70a0a0b3900aac5aee704ae8') );

    # set the payload to be the base 64 encoded string representation of the file
    push(@params, SOAP::Data->name(payload => $base64) );

    # invoke the web service
    my $result = $soap->call($method => @params);

    print_fault($result);

    print_result($result);
}

sub print_result {
    my ($result) = @_;

    if ($result->body && $result->body->{'insertResponse'}) {
        my %keyHash = %{ $result->body->{'insertResponse'} };
        foreach my $k (keys %keyHash) {
            print "name=$k    value=$keyHash{$k}\n";
        }
    }
}

sub print_fault {
    my ($result) = @_;

    if ($result->fault) {
```

```
    print "faultcode=" . $result->fault->{'faultcode'} . "
\n";
    print "faultstring=" . $result->fault->{'faultstring'}
} . "\n";
    print "detail=" . $result->fault->{'detail'} . "\n";
}
}

# use the itil user for basic auth credentials
sub SOAP::Transport::HTTP::Client::get_basic_credentials {
    return 'itil' => 'itil';
}
```

Override a SOAP endpoint

The SOAP endpoint address where the SOAP message is posted is consistent with the endpoint of the WSDL.

In some cases, however, the WSDL may reference an incorrect endpoint URL. If this happens, it is necessary to over-ride the generated URL by creating the property com.glide.soap_address_base_url to contain the new URL. You may have to add the property manually as this is not an base system property.

For instance, a generated SOAP endpoint may look like this:

```
https://instance.service-now.com/incident.do?SOAP
```

You can specify a property to define the endpoint such that it goes through a proxy by setting the property:

```
com.glide.soap_address_base_url = "https://myproxy.mycompany.com/service-now/"
```

This will result in the endpoint being generated to appear as:

```
https://myproxy.mycompany.com/service-now/incident.do?SOAP
```

Enable HTTP compression

By default, the SOAP request is accepted un-compressed and the result of the request is returned un-compressed.

To enable HTTP compression using [gzip] when sending in your SOAP request, set the following HTTP header:

Content-Encoding: gzip

To receive the SOAP response compressed using [gzip] send in your SOAP request with the following HTTP header:

Accept-Encoding: gzip

Prevent empty elements in SOAP messages

By default, an instance does not omit empty elements, elements with NULL or NIL values, from SOAP messages.

To prevent SOAP responses from containing empty elements, an administrator can create a system property called `glide.soap.omit_null_values` and set the value to **true**. This behavior is compliant with the WSDL as all elements in a SOAP message have a `minOccurs=0` attribute and are therefore optional. In addition, this behavior prevents the instance from creating inefficient SOAP messages containing a large number of empty elements.

Set this property to **false** to allow SOAP messages to search for existing fields with empty values. For example, if an administrator wants to search for incidents with an empty **Assigned to** field from a SOAP message, then the SOAP message must be able to send an empty value for this field.

Note: Changing the value of this property may cause unintended actions in existing web service integrations. Administrators are strongly encouraged to carefully test the new behavior to ensure that existing integrations process empty elements as expected.

Insert related records using SOAP

Support is available for inserting hierarchical data into tables or web service import set tables. The hierarchical data in the Insert API is automatically mapped to related records of the targeted table.

Before you begin

Role required: admin

Procedure

Create and set the property `glide.web_service.hierarchical` to true.

The client of the API can also override this value by invoking the SOAP web service with the URL parameter hierarchical=true.

Example

For example, when a related list is created for the incident table called u_custom_comments:

Hierarchical Incident

The screenshot shows the ServiceNow interface for a hierarchical incident. At the top, there is a form for an incident with fields: Number (INC0010001), Opened (2010-03-21 21:30:33), Caller, Closed, Short description (test hierarchical), and Additional comments. Below the form is a related list titled "Custom Comments" for the incident. The list shows one item: comment 1, which is of type travel. The comment text is "comment 1".

And u_comment_items is created as a related list for u_custom_comments:

Hierarchical Custom Comments

The screenshot shows the ServiceNow interface for a hierarchical custom comment. At the top, there is a form for a custom comment with fields: Comment (comment 1), Comment Type (travel), and Incident (INC0010001). Below the form is a related list titled "Comment Items" for the comment. The list shows one item: value 1, which is named name 1.

- WSDL Schema with related records

When a WSDL for the target Incident table is requested with an additional parameter of hierarchical=true, the WSDL schema for the Insert function will reflect available related records that may participate in the hierarchical data payload.

- [Hierarchical SOAP Message](#)

When the SOAP message is constructed from the hierarchical web service described by the WSDL and invoked, it will create the `incident`, `u_custom_comments`, and `u_comment_items` records.

Related reference

- [WSDL Schema with related records](#)
- [Hierarchical SOAP Message](#)

When a WSDL for the target Incident table is requested with an additional parameter of hierarchical=true, the WSDL schema for the Insert function will reflect available related records that may participate in the hierarchical data payload.

For example, the insert portion of the schema of the incident table, when requested with hierarchical=true displays its hierarchy as follows:

<https://instance.service-now.com/incident.do?WSDL&hierarchical=true>

WSDL Schema

```
<xsd:element name="insert">
  - <xsd:complexType>
    - <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="active" type="xsd:boolean"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="activity_due" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="approval" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="approval_history" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="approval_set" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="assigned_to" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="assignment_group" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="business_duration" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="business_stc" type="xsd:integer"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="calendar_duration" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="calendar_stc" type="xsd:integer"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="caller_id" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="category" type="xsd:string"/>
    ...
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

WSDL Schema Continued

```
<xsd:element maxOccurs="1" minOccurs="0" name="watch_list" type="xsd:string"/>
<xsd:element maxOccurs="1" minOccurs="0" name="work_end" type="xsd:string"/>
<xsd:element maxOccurs="1" minOccurs="0" name="work_notes" type="xsd:string"/>
<xsd:element maxOccurs="1" minOccurs="0" name="work_start" type="xsd:string"/>
- <xsd:element minOccurs="0" name="u_custom_comments">
  - <xsd:complexType>
    - <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="u_comment" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="u_comment_type" type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="u_incident" type="xsd:string"/>
    - <xsd:element minOccurs="0" name="u_comment_items">
      - <xsd:complexType>
        - <xsd:sequence>
          <xsd:element maxOccurs="1" minOccurs="0" name="u_comment" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="u_name" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="u_value" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

The WSDL above shows the incident table having a related table `u_custom_comments` that itself has a related table `u_comment_items`.

When the SOAP message is constructed from the hierarchical web service described by the WSDL and invoked, it will create the `incident`, `u_custom_comments`, and `u_comment_items` records.

Endpoint URL

<https://instance.service-now.com/incident.do?SOAP&hierarchical=true>

Request

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  xmlns:inc="http://www.service-now.com/incident">
  <soapenv:Header/>
  <soapenv:Body>
    <inc:insert>
      <short_description>test hierarchical</short_description>
    </inc:insert>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<u_custom_comments>
    <u_comment>comment 1</u_comment>
    <u_comment_type>travel</u_comment_type>
    <u_comment_items>
        <u_name>name 1</u_name>
        <u_value>value 1</u_value>
    </u_comment_items>
</u_custom_comments>
</inc:insert>
</soapenv:Body>
</soapenv:Envelope>
```

Reponse

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:inc="http://www.service-now.com/incident">
    <soapenv:Header/>
    <soapenv:Body>
        <insertResponse>
            <sys_id>8422ebe7c0a8006e7d23848c2dc8ba47</sys_id>
            <number>INC0010001</number>
        </insertResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Specify requirement for signed SOAP requests

Use a SOAP security policy to specify whether the instance requires signed SOAP requests for all inbound SOAP traffic.

Before you begin

Role required: web_service_admin or admin

About this task

By default, all inbound SOAP traffic must be signed. Administrators may want to disable this policy and allow unsigned SOAP requests to ServiceNow web services.

Procedure

1. Navigate to **All > System Web Services > SOAP Security Policies**.
2. Select the **Default Policy**.
3. Clear the **Required to Sign SOAP Request** check box (selected by default) to disable the requirement.
4. Click **Update**.

Activate the Enhanced Web Service Provider - Common plugin

Administrators can activate the Enhanced Web Service Provider - Common plugin to enable unsigned WS-Security requests and specify what authentication requirements SOAP requests have.

Before you begin

Role required: admin

Procedure

1. Navigate to **All > System Applications > All Available Applications > All**.
2. Find the Enhanced Web Service Provider - Common plugin (`com.glide.web_service_provider_v2`) using the filter criteria and search bar.

You can search for the plugin by its name or ID. If you cannot find a plugin, you might have to request it from ServiceNow personnel.

3. Select **Install**, and then in the Activate Plugin dialog box, select **Activate**.

Note: When domain separation and delegated admin are enabled in an instance, the administrative user must be in the **global** domain. Otherwise, the following error appears: Application installation is unavailable because another operation is running: Plugin Activation for <plugin name>.

- [Installed with the Enhanced Web Service Provider - Common plugin](#)

The following components installed with the Enhanced Web Service Provider - Common plugin.

The following components installed with the Enhanced Web Service Provider - Common plugin.

The Enhanced Web Service Provider - Common plugin installs the following components:

Components installed with the Enhanced Web Service Provider - Common plugin

Component Type	Component Installed	Description
Module	Web Services Security Profiles	The plugin adds this module to the System Web Services application.
System Property	<code>glide.soap.default_security_policy</code>	Specifies the default security policy to use when enforcing Web Services-Security (WSS) for inbound requests.

Configure SOAP security

Administrators can configure web service security for inbound SOAP requests made to the ServiceNow instance.

Before you begin

Role required: admin

About this task

You can also set up web service security to use different certificates for different web service clients. By enabling web service security, you can prevent man-in-the-middle attacks.

Note: After you configure a WS-security profile or a security policy, validation is performed on all incoming SOAP requests, including from the MID Server or ODBC driver. Disable validation for these types of requests by marking the service accounts as internal integration users.

Procedure

1. [Upload a certificate to an instance.](#)
2. [Create a WS-security profile.](#)
3. [Create a security policy.](#)
Security policies define which WS-security profiles are used to evaluate a particular web service request. If no policy is defined, all WS-security profiles are used to evaluate all requests.
4. Set the value of the property `glide.soap.default_security_policy` to the name of the new security policy.

Set the SOAP default security policy

Set the SOAP default security policy.

Before you begin

Role required: `web_service_admin` or `admin`

Procedure

1. Navigate to **All > System Web Services > Properties**.
2. In the **Security Policy to enforce if WS-Security is enabled** field, enter the default security policy to use when enforcing WS-security.
3. Click **Save**.

Create a new security policy

Administrators can specify which security profiles WS-Security communications must meet by creating a new security policy.

Before you begin

Role required: web_service_admin or admin

Procedure

1. Navigate to **All > System Web Services > SOAP Security Policies**.
2. Click **New**.
3. Fill out the SOAP Security Policy form (see table).

SOAP Security Policy form fields

Field	Description
Name	Enter a unique name for the security policy. Use this name to set the default security policy with the <code>glide.soap.default_security_policy</code> property.
Type	Select whether the SOAP security policy applies to inbound or outbound traffic.
Required to Sign SOAP Request	Select this checkbox to require signed SOAP requests. Clear the checkbox to allow unsigned SOAP requests. When enabled, the instance will produce an error for any SOAP request that does not include a valid signature. When disabled, the instance still verifies any signature included with a SOAP request.
Authenticate	Select if a SOAP request must authenticate against all security profiles or at least one security profile.

Field	Description
Security Profiles	Select the security profiles you want to apply to this SOAP security policy. You must select at least one security profile. For more information, see WS-Security .

4. Click **Submit**.

Create a new WS-Security profile

Create a new WS Security profile to define how to authenticate a web services message when WS-Security is enabled.

Before you begin

Role required: web_service_admin or admin

Procedure

1. Navigate to **All > System Web Services > WS Security Profiles**.
2. Click **New**.
3. Fill in the WS-Security Profile form (see table).

WS-Security Profile form fields

Field	Description
Name	Enter a unique name for the security profile.
Type	Select X509 to verify the request's certificate. Select Username to verify the request's user credentials.
Run as user	Select the ServiceNow user the instance will impersonate if authentication succeeds and

Field	Description
	the Bind Session field is selected. All web services requests will be attributed to this user. For example, if you select the System Administrator user then the instance treats all web services operations as being done by the system administrator. Make sure the user you select has appropriate SOAP privileges if you are using the <code>glide.soap.strict_security</code> high security setting. This field is only visible when the type is X509 .
Order	Enter the order in which the instance checks security profiles. The instance checks all security profiles when processing an incoming SOAP request. If a request fails any security profile authentication requirement, the instance stops processing additional security profiles and fails the request.
Bind Session	Select this check box to have the instance impersonate the user listed in the Run as user field. You should only set this field for one profile at a time. If multiple profiles have this field selected, ServiceNow impersonates the user from the last successfully authenticated WS-Security profile. If no profile has this field selected, ServiceNow impersonates the user provided with the basic authentication headers or

Field	Description
	impersonates the default user (Guest).
X509 Certificate	Select the certificate record containing the certificate for web service requests. ServiceNow only validates the request signature. It automatically trusts the certificate's certificate authority (CA). This field is only visible when the type is X509 .
Profile action	Select how you want the instance to authenticate the user credentials. Select Authenticate with user if you want the instance to authenticate the request based on an existing user record. The request's credentials must match values in an existing user record. Select Specify user to authenticate if you want to list a user name and password combination that all web services requests must meet. The request's credentials must match the user name and password you list. This field is only visible when the type is Username .
User field to match UserName	Select the column from the User [sys_user] table containing the value you want match against the request's UserName. For example, if you select Email then the request UserName header must contain an email address matching an existing ServiceNow

Field	Description
	user. This field is only visible when the profile action is Authenticate with user .
User name	Enter the user name that all web services requests must contain. This field is only visible when the profile action is Username .
User password	Enter the password that all web services requests must contain. This field is only visible when the profile action is Username .

The form is titled "WS Security Profile". It contains fields for Name (Username), Type (Username), Bind session (checked), Profile action (Authenticate with user), Order (100), and User field to match UserName (Email). A "Submit" button is at the bottom.

4. Click **Submit**.

Related topics

- [SOAP web services security](#)
- [WS-Security](#)
- [WS-Security profiles](#)

Enforce strict security for inbound SOAP

Strict security for web services requires that users meet Contextual Security requirements to access instance resources.

Before you begin

Role required: admin

About this task

Note: ServiceNow does not support digital certificates, digital signatures, or other digested token methods in SOAP web service calls.

To enforce strict security for web services connections:

Procedure

1. Navigate to **All > System Properties > Web Services**.
2. Select **Yes** for **Enforce strict security on incoming SOAP requests**.

Note: To learn more about this property, see [SOAP request strict security \(instance security hardening\)](#) in Instance Security Hardening Settings.

Enable WS-Security verification

Administrators can enable Web Services Security (WSS) verification from the Web Services system properties.

Before you begin

Role required: web_service_admin or admin

Procedure

1. Navigate to **All > System Web Services > Properties**.
2. For **Require WS-Security header verification for all incoming SOAP requests**, select **Yes**.

Require WS-Security header verification for all incoming SOAP requests
 Yes | No

Note: Selecting this option enables WS-Security for all inbound SOAP requests. It is not possible to enable WS-Security for only some requests.

3. Click **Save**.

4. Create a [WS-security profile](#).
5. Update the user record for the [Mid Server](#) and [ODBC driver](#) to mark [these users as internal integration users](#).
6. Download and install the latest [MID Server](#) and [ODBC driver](#).
7. To validate SOAP request signatures, upload the remote web service's certificate as a JKS and create the web service's WSS Username Token Profile.

Note: Because ServiceNow WSS implementation does not verify the CA certificate, you do not need to upload the web service's CA certificate.

- [Mark service accounts as internal integration users](#)

Allow internal integration communications to bypass the WSS authentication requirement by marking their user accounts as internal integration users.

Related topics

- [Basic authentication](#)

Allow internal integration communications to bypass the WSS authentication requirement by marking their user accounts as internal integration users.

Before you begin

Role required: admin

About this task

When WS-Security is enabled, authentication is required for all SOAP requests including internal integration communications such as the MID Server, ODBC Driver, Remote Update Sets, and high availability cloning. SOAP requests for these internal integration communications cannot implement WS-Security due to technical implications. If your instance uses these SOAP interfaces, you can allow them to bypass the WS-Security authentication requirement by marking their user accounts as internal integration users.

Note: Any web services other than ODBC, MID Server, Remote Update Sets, or high availability cloning must implement WS-Security headers when WS-Security is enabled.

Procedure

1. Navigate to **All > User Administration > Users**.
2. Select the user account for the MID Server or ODBC Driver.
3. Configure the form to add the **Internal Integration User** field.
4. Select the **Internal Integration User** check box.
5. Click **Update**.

Related tasks

- [Enable WS-Security verification](#)

Related topics

- [WS-Security](#)

Debug incoming SOAP envelope

To capture incoming SOAP envelope XML in the system log, add the property `glide.processor.debug.SOAPProcessor` with a value of `true`.

When enabled, this property adds the incoming SOAP envelope in the **Message** field of the system log (**System Logs > All**). Disable this debugging feature as soon as you are finished so that the log is not overwhelmed with excessive and unnecessary debugging information.

View a SOAP session log

You can view a user's log from a SOAP session.

Before you begin

Role required: admin

Procedure

1. Navigate to All > User Administration > Logged in users.

Active	User	Last transaction	Transaction duration	Last accessed	Last transaction time	Total transactions	Locked
<input checked="" type="checkbox"/> true	admin	https://demosw.service-now.com/ecc_queue...	0:00:00.060	2011-04-25 14:45:00	2011-04-25 14:45:00	7,314	false
<input checked="" type="checkbox"/> false	admin	https://demosw.service-now.com/ecc_queue...	0:00:00.032	2011-04-25 14:44:39	2011-04-25 14:44:54	6,379	false
<input checked="" type="checkbox"/> true	admin	/v_user_session_list.do	0:00:00.037	2011-04-25 14:44:53	2011-04-25 14:45:00	56	false
Actions on selected rows...							

Active SOAP Session

Response time(ms): 665, network: 274, browser: 356, server: 35

2. Open an active SOAP session to see the transactions log.

The SOAP session is marked as inactive within 60 seconds of the last transaction.

Basic authentication code samples

Samples of basic authentication code for several programming languages and versions.

Perl and the SOAP::Lite libraries

To supply basic authentication when using Perl and the SOAP::Lite libraries, you can implement the following function:

```
sub SOAP::Transport::HTTP::Client::get_basic_credentials {
    return 'user_name' => 'password';
}
```

C# .NET VS 2005 or older

When using C# .NET VS 2005 or older, you can take advantage of the Credentials object. For example:

```
System.Net.ICredentials cred = new System.Net.NetworkCredential("user_name", "password");

service.ServiceNow proxy = new service.ServiceNow();
service.get getService = newservice.get();
service.getResponse getServiceResponse = new service.getResponse();

try
```

```
{  
    proxy.Credentials = cred;  
    getService.sys_id = "bf522c350a0a140701972dbf876f1610";  
    getServiceResponse = proxy.get(getService);  
    catch (Exception ex) { }  
}
```

C# .NET VS 2008

When using C# .NET VS 2008, you can take advantage of the ClientCredentials object. For example:

```
Demo_Incident.ServiceNowSoapClient client = new Test08Webs  
ervice.Demo_Incident.ServiceNowSoapClient();  
client.ClientCredentials.UserName.UserName = "admin";  
client.ClientCredentials.UserName.Password = "admin";
```

Then in your app.config file look for the following and change "None" to "Basic":

```
<transport clientCredentialType="None" proxyCredentialType  
="None" realm="" />
```

VB .NET

When using VB .NET taking advantage of the Credentials object would look like the following:

```
Sub Main()  
    Dim cred As New System.Net.NetworkCredential("user  
_name", "password")  
  
    Dim proxy As New VB_Democm.incident.ServiceNow  
    Dim getIncident As New VB_Democm.incident.get  
    Dim getResponse As New VB_Democm.incident.getRespo  
nse  
  
    proxy.Credentials = cred  
  
    getIncident.sys_id = "[your sysID here]"  
  
    getResponse = proxy.get(getIncident)  
  
End Sub
```

The resulting response when Basic Authentication is turned on and no credentials are supplied looks like this:

```
<html>
<head>
<title>Apache Tomcat/5.0.28 - Error report</title>
<style>  <!--H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;}>
    H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;}>
    H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;}>
    BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;}>
    B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;}>
    P {font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;}>
    A {color: black;}>
    A.name {color: black;}>
    HR {color: #525D76;}-->
</style>
</head>
<body>
<h1>HTTP Status 401 -</h1>
<HR size="1" noshade="noshade">
<p><b>type</b> Status report</p>
<p><b>message</b> <u></u></p>
<p><b>description</b>
<u>This request requires HTTP authentication ()</u></p>
>
<HR size="1" noshade="noshade">
<h3>Apache Tomcat/5.0.28</h3>
</body>
</html>
```

Example WS-Security SOAP envelope header

An example of a valid WS-Security SOAP envelope header.

```
<SOAP-ENV:Header>
  <wsse:Security
    xmlns:wsse="http://docs.oasis-open.org/wss
```

```
/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    SOAP-ENV:mustUnderstand="1">
        <wsse:BinarySecurityToken
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
                wsu:Id="CertId-2D914AB929A6719E7F13068829874641"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
                MII EgzCCA2ugAwIBAgILAQAAAAABLOZQMtEwDQYJKoZIhvcNAQdCBJbmMxJTAjBgbnRpYWwgQ0EwgyMjU1WjB3MQswIjZS1Ob3cxKDAxJ2aNlW5vdy5jbhcn3uZXIgRGV2ZWx4IBDwAwggEKAoIBbTDUvw2nG1NA9y9iEVcwi/rZcbEPj8oWyLVAKkc64WC7Y4HpBdfW49ff92TKD7MN3Vkyhz2utc1gjCy0U+f0NXKgh0Q
            EFBQAwQDEXMBUGA1UEChMOQ3lizZXJ0cnVzNVBAMTHEN5YmVydHJ1c3QgU3VyzWNyzWRlHhcNMTAxMjE0MTgyMjU1WhcNMTECMjE0MTcQYDVQQGEwJVUzEUMBIGA1UEChMLU2VydmmbGkqhkiG9w0BCQEWSWRhdmlkLmxvb0BzM20xKDAmBgNVBAMTH1NlcnZpY2UtTm93IEBvcG1lbnQwggEiMA0GCSqGSIb3DQEBAQUAAQCVtcRIB6zkGnN9uyhtcSDNSIuCW6FgnTwTp5TG3eELOOFBCuRLLeY5x281N+cJ72v+zOqJThgrzhDabj0vDM/zU8bvAXcw6FoCUDFJT7FBgDQ3LEudq80Up+TfETiGwrEA3jRgyxZLOFiN5HJixl9juNjmLWugqqIG04yZSuCrRheNpwCqWbbJvLbR9Ybs0613UAYCQ09hr
```

```
RnI7VaPvfiueUvuLopap
AGjggFFMIIBQTAfBgNVH
C1TA7BgnVHR8ENDA
290LmNvbS9TdxJ1Q3J1Z
cdiYmq0enW6mgaV
VR0PAQH/BAQD
3rBgEFBQcBA1Q
WNlcnQub21uaXJv
AkBgNVHREEHTAbg
tMB0GA1UdJQQWMBQG
iG9w0BAQUFAAO
aHP5wTxqVlFxzJy1zi6
E8QCETkjYNyuWJd9zm
AMrH1/H673E4myzvU
klyZaKtMp+31qmf3bG
qcNhgw6V1AWG566g
2v7f44OSekJeBvTfZCR
WA==

</wsse:BinarySecurityToken>
<ds:Signature xmlns:ds="http://www.w3.org/
2000/09/xmldsig#" Id="Signature-2">
<ds:SignedInfo xmlns:ds="http://ww
w.w3.org/2000/09/xmldsig#">
<ds:CanonicalizationMethod
Algorithm="http://
www.w3.org/2001/10/xml-exc-c14n#" xmlns:ds="http://www.w3.
```

```
org/2000/09/xmldsig#" />
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#" />
        <ds:Reference URI="#Timest
amp-1" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:Transforms xml
ns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:Transf
orm Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" xm
ns:ds="http://www.w3.org/2000/09/xmldsig#" />
                </ds:Transforms>
                <ds:DigestMethod A
lgorithm="http://www.w3.org/2000/09/xmldsig#sha1"
                    xmlns:ds="
http://www.w3.org/2000/09/xmldsig#" />
                    <ds:DigestValue xm
ns:ds="http://www.w3.org/2000/09/xmldsig#">NIS5sizg8wttGL
+aWFQ4003TpPg=</ds:DigestValue>
                </ds:Reference>
                <ds:Reference URI="#id-3"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                    <ds:Transforms xml
ns:ds="http://www.w3.org/2000/09/xmldsig#">
                        <ds:Transf
orm Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" xm
ns:ds="http://www.w3.org/2000/09/xmldsig#" />
                        </ds:Transforms>
                        <ds:DigestMethod A
lgorithm="http://www.w3.org/2000/09/xmldsig#sha1"
                            xmlns:ds="
http://www.w3.org/2000/09/xmldsig#" />
                            <ds:DigestValue xm
ns:ds="http://www.w3.org/2000/09/xmldsig#">/rXB+nhBT5BXtD
EriUIBOyhoh8Y=</ds:DigestValue>
                        </ds:Reference>
                    </ds:SignedInfo>
                    <ds:SignatureValue xmlns:ds="http:
//www.w3.org/2000/09/xmldsig#">
                        fwjxJRiDNrNxbVsKoHZflsmKLY
ADldJf0BoN3R2Fx9rjpszFXI2Gp92eXsP+S16rmbPXIdKb81L1
                    </ds:SignatureValue>
                </ds:Reference>
            </ds:Transforms>
        </ds:Signature>
    </ds:SignedInfo>
</ds:Signature>
```

```
+dv8up18WYPrKJP61KeJ0ZsKND
X474NYC2XEzdJcXbZNktmqY0dSmKwJZzi8rJtmGrbOUAaH51GK
          oXV2FLJ0AqILOZMyP/SPWKbOUN
UCpssY7vRA+tX8ZmrjTwMUvpOZbo+KInPmwfpZ6n/uarOh5zjL
          NaYJylTCjuuqXDKPZLvdqy48yr
sGAWczB901KwLLrE8C+6aPucFrTBytX91vIhaWiLZuba8Nouaz
          vUkjUk7LM5o87MGrSFx30wxbaO
D7/cMtgd2bxA==
          </ds:SignatureValue>
          <ds:KeyInfo Id="KeyId-2D914AB929A6
719E7F13068829875022"
          xmlns:ds="http://www.w3.or
g/2000/09/xmldsig#">
          <wsse:SecurityTokenReferen
ce
          xmlns:wsu="http://
docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurit
y-utility-1.0.xsd"
          wsu:Id="STRId-2D91
4AB929A6719E7F13068829875053"
          xmlns:wsse="http:/
/docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
ty-secext-1.0.xsd">
          <wsse:Reference UR
I="#CertId-2D914AB929A6719E7F13068829874641"
          ValueType=
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x
509-token-profile-1.0#X509v3"
          xmlns:wsse
="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" />
          </wsse:SecurityTokenRefere
nce>
          </ds:KeyInfo>
          </ds:Signature>
          <wsu:Timestamp
          xmlns:wsu="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xs
d"
          wsu:Id="Timestamp-1">
          <wsu:Created>2011-05-31T23:03:07.4
54Z</wsu:Created>
          <wsu:Expires>2011-05-31T23:08:07.4
54Z</wsu:Expires>
```

```
</wsu:Timestamp>
<wsse:UsernameToken>
    <wsse:Username>test_user</wsse:User
rname>
    <wsse:Password>xxxxxx</wsse:Passwo
rd>
    </wsse:UsernameToken>
</wsse:Security>
</SOAP-ENV:Header>
```

WS-Security properties

These properties control the behavior of WS-Security X.509 tokens.

Properties

Property	Description
glide.soap.msg_digest.algorithm	<p>Specifies the method digest algorithm. Possible values are SHA-1, SHA-256, and SHA-512.</p> <ul style="list-style-type: none">• Type: String• Default value: SHA-1• Location: Add to the System Property [sys_properties] table
glide.soap.signature.algorithm	<p>Specifies the signature algorithm. Possible values are RSA-SHA-1, RSA-SHA-256, and RSA-SHA-512.</p> <ul style="list-style-type: none">• Type: String• Default value: RSA-SHA-1• Location: Add to the System Property [sys_properties] table

Property	Description
glide.soap.canonical.algorithm	<p>Specifies the canonicalization algorithm. Possible values are Canonical xml 1.0, Canonical xml 1.0 with Comments, Exclusive Canonical xml 1.0, and Exclusive Canonical xml 1.0 with Comments.</p> <ul style="list-style-type: none">• Type: String• Default value: Exclusive Canonical xml 1.0• Location: Add to the System Property [sys_properties] table

Each possible value for these properties represents a standard WS-Security algorithm.

Property value URLs

Value	Algorithm
Method digest algorithms	
SHA-1	http://www.w3.org/2000/09/xmldsig#sha1
SHA-256	http://www.w3.org/2001/04/xmlenc#sha256
SHA-512	http://www.w3.org/2001/04/xmlenc#sha512
Signature algorithms	
RSA-SHA-1	http://www.w3.org/2000/09/xmldsig#rsa-sha1
RSA-SHA-256	http://www.w3.org/2001/04/xmldsig-more#rsa-sha256

Value	Algorithm
RSA-SHA-512	http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
Canonicalization algorithms	
Canonical xml 1.0	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Canonical xml 1.0 with Comments	http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
Exclusive Canonical xml 1.0	http://www.w3.org/2001/10/xml-exc-c14n#
Exclusive Canonical xml 1.0 with comments	http://www.w3.org/2001/10/xml-exc-c14n#WithComments

WS-Security error messages

An instances produces one of the following error messages when it encounters an issue with a WS-security SOAP message.

WS-security error messages

Error	Description
Invalid Security Policy Selected. Select an Inbound policy for Inbound Requests.	The default policy name is set to an outbound policy. Set the default policy name to an inbound security policy.
SOAP request not Signed. Policy requires signing.	The SOAP security policy requires signing and the inbound SOAP request is not signed. Either specify a different SOAP security policy or provide the SOAP request with a proper signature.

Error	Description
No profiles to authenticate.	The selected Security policy either does not have any security profiles (X509 or UserNameToken) or the security profiles are inactive. Verify at least one security profile is active.
Unable to verify signed request.	The inbound SOAP request contains an invalid signature. The SOAP request changed after being signed.
Failed to extract principal(s) from request.	The SOAP request has either been tampered or was not well formed. ServiceNow cannot extract credentials to authenticate the request.
Failed to authenticate WS-security, unknown type.	The SOAP request contains an unsupported security profile. Resend the request with a supported security profile type : X509 or UsernameToken.
Failed to authenticate WS-security.	ServiceNow failed to authenticate against the provided profile credentials. Verify that the SOAP request is using the proper credentials.

LongRunningSOAPRequestProps

The following properties are available for long-running SOAP requests.

Long-running SOAP request properties

Property	Description
glide.http.connection_timeout	Specify the maximum number of milliseconds an outbound HTTP

Property	Description
	<p>request (such as Web Services) will wait to establish a connection.</p> <ul style="list-style-type: none">• Type: integer• Default value: 10000 (10 seconds)• Location: system properties [sys_properties] table
glide.http.timeout	<p>Specifies the maximum number of milliseconds to wait before an outbound transaction times out.</p> <ul style="list-style-type: none">• Type: integer• Default value: 175000 (175 seconds)• Location: Add to system properties [sys_properties] table
glide.soap.max_redirects	<p>Specifies the maximum number of redirects the server sends to the client before the soap request is timed out.</p> <ul style="list-style-type: none">• Type: integer• Default value: 20• Location: system properties [sys_properties] table
glide.soap.request_processing_timeout	<p>Specify the maximum number of seconds an inbound SOAP request has to finish processing before the connection times out. This property computes a default value</p>

Property	Description
	<p>from the value of the property <code>glide.http.timeout</code> divided by 1000.</p> <p>This property accepts values 5–1200 seconds (20 minutes).</p> <p>Customers might have network infrastructure (such as proxy servers) in place which implement a shorter timeout. In this case, a socket timeout may occur unless this property is set to a shorter value. Set this property to a value several seconds less than the shortest socket inactivity timeout in effect anywhere in the network path between the client application and the instance.</p> <ul style="list-style-type: none">• Type: integer• Default value: 60• Location: system properties [sys_properties] table
<code>glide.soapprocessor.allow_long_running_threads</code>	Enables or disables a 307-Temporary Redirect when the request includes a <code>redirectSupported=true</code> parameter. The default setting is true (enabled).
<code>glide.soapprocessor.max_long_running_threads</code>	Controls the maximum number of long-running SOAP threads which can run at any one time. The default value for this property is determined dynamically based on the number of SOAP semaphores configured. It should not be necessary to change this value.

JSONv2 web service

The ability to describe sets of data in JSON format is a natural extension to the JavaScript language.

ServiceNow supports a web service interface that operates on the JSON object as the data input and output format.

The JSONv2 web service is provided by a platform-level processor similar to the services for SOAP, WSDL, CSV, Excel, and XML. Like those services, the JSON web service is triggered by a standalone JSONv2 URL parameter. For example:

```
https://<instance name>.service-now.com/mytable.do?JSONv2
```

Having the JSON object available as a data format for web services means that you can create (insert), update, and query any data in the system using the JSON object format, and get results in the JSON object format.

Security

Like all other HTTP-based web services available on the platform, the JSONv2 web service is required to authenticate using basic authentication by default. The user ID that is used for authentication is subjected to access control in the same way as an interactive user.

- [JSON object format](#)

The JSON object is built in two structures.

- [JSON response status](#)

JSONv2 requests may return one of several response statuses.

- [Setting the number of rows returned](#)

The following system property controls how many rows JSON returns with each query.

- [Requiring basic authentication for incoming JSONv2 requests](#)

The following system property controls whether basic authentication is required for incoming JSONv2 requests.

- [Action parameters](#)

Action parameters are separate and different from data parameters because they specify the action to take when the JSON object parameter is part of an HTTP GET or POST request.

- [JSON Data Retrieval API](#)

Query for data by issuing an HTTPS GET request to the instance.

- [JSON Data Modification API](#)

Modify data using the JSON web service by sending an HTTPS POST request to the instance.

JSON object format

The JSON object is built in two structures.

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

In its simplest form, a JSON object is just a comma delimited set of name/value pairs. For example:

```
{"name one": "value one", "name two": "value two"}
```

The following is a sample of a single record array of incidents in JSON:

```
{"records": [{"closed_by": "",  
    "_status": "success",  
    "category": "inquiry",  
    "escalation": "0",  
    "state": "1",  
    "location": "",  
    "reassignment_count": "0",  
    "time_worked": "",  
    "order": "0",  
    "due_date": "",  
    "number": "INC0010180",
```

```
"upon_approval":"proceed",
"sla_due":"2010-03-04 22:51:49",
"follow_up":"",
"notify":"1",
"business_stc":"0",
"caused_by":"",
"rejection_goto":"",
"assignment_group":"d625dccec0a8016700a222a0f7900d06",
"incident_state":"1",
"opened_at":"2010-02-23 22:51:49",
"wf_activity":"",
"calendar_duration":"",
"group_list":"",
"caller_id":"",
"comments":"",
"priority":"3",
"sys_id":"fd0774860a0a0b380061bab9094733ad",
"sys_updated_by":"itil",
"variables":"",
"delivery_task":"",
"sys_updated_on":"2010-02-23 22:51:49",
"parent":"",
"active":"true",
"opened_by":"681b365ec0a80164000fb0b05854a0cd",
"expected_start":"",
"sys_meta":"System meta data",
"watch_list":"",
"company":"",
"upon_reject":"cancel",
"work_notes":"",
"sys_created_by":"itil",
"cmdb_ci":"",
"approval_set":"",
"user_input":"",
"sys_created_on":"2010-02-23 22:51:49",
"contact_type":"phone",
"rfc":"",
"approval_history":"",
"activity_due":"",
"severity":"3",
"subcategory":"",
"work_end":"",
"closed_at":"",
"close_notes":",
```

```
        "variable_pool":"",
        "business_duration":"",
        "knowledge":"false",
        "approval":"not requested",
        "sys_mod_count":"0",
        "problem_id":"",
        "calendar_stc":"0",
        "work_start":"",
        "sys_domain":"global",
        "sys_response_variables":"",
        "correlation_id":"",
        "sys_class_name":"incident",
        "short_description":"this was inserted with python",
        "impact":"1",
        "description":"",
        "correlation_display":"",
        "urgency":"3",
        "assigned_to":"",
        "made_sla":"true",
        "delivery_plan":""}
    ]
}
```

The following is a record array of incident responses with an error.

```
{
  "records": [
    {
      "__error": {
        "message": "Invalid Insert into: incident"
      },
      "reason": "Data Policy Exception: Short description is mandatory "
    },
    {
      "__status": "failure",
      "active": "true",
      "activity_due": "",
      "approval": "not requested",
      "approval_history": "",
      "approval_set": "",
      "assigned_to": "",
      "assignment_group": "d625dccec0a8016700a222a0f7900d06",
      "business_duration": "",
      "business_stc": ""
    }
  ]
}
```

```
    "calendar_duration": "",  
    "calendar_stc": "",  
    "caller_id": "",  
    "category": "inquiry",  
    "caused_by": "",  
    "child_incidents": "0",  
    "close_code": "",  
    "close_notes": "",  
    "closed_at": "",  
    "closed_by": "",  
    "cmdb_ci": "",  
    "comments": "",  
    "comments_and_work_notes": "",  
    "company": "",  
    "contact_type": "phone",  
    "correlation_display": "",  
    "correlation_id": "",  
    "delivery_plan": "",  
    "delivery_task": "",  
    "description": "",  
    "due_date": "",  
    "escalation": "0",  
    "expected_start": "",  
    "follow_up": "",  
    "group_list": "",  
    "impact": "3",  
    "incident_state": "1",  
    "knowledge": "false",  
    "location": "",  
    "made_sla": "true",  
    "notify": "1",  
    "number": "INC0010001",  
    "opened_at": "2013-07-23 18:01:17",  
    "opened_by": "6816f79cc0a8016401c5a33be04be441",  
    "order": "",  
    "parent": "",  
    "parent_incident": "",  
    "priority": "5",  
    "problem_id": "",  
    "reassignment_count": "0",  
    "reopen_count": "0",  
    "resolved_at": "",  
    "resolved_by": "",
```

```
        "rfc": "",  
        "severity": "3",  
        "short_description": "",  
        "skills": "",  
        "sla_due": "",  
        "state": "1",  
        "subcategory": "",  
        "sys_class_name": "incident",  
        "sys_created_by": "admin",  
        "sys_created_on": "2013-07-23 18:01:17",  
        "sys_domain": "global",  
        "sys_id": "a96479343cb60100a92ec9a477ba9e45",  
        "sys_mod_count": "0",  
        "sys_updated_by": "admin",  
        "sys_updated_on": "2013-07-23 18:01:17",  
        "time_worked": "",  
        "upon_approval": "proceed",  
        "upon_reject": "cancel",  
        "urgency": "3",  
        "user_input": "",  
        "watch_list": "",  
        "work_end": "",  
        "work_notes": "",  
        "work_notes_list": "",  
        "work_start": ""  
    }  
]  
}
```

JSON response status

JSONv2 requests may return one of several response statuses.

JSON Success Response

Each JSON success response includes a record array containing the records retrieved by the given action. Each JSON object contains one or more metadata elements, prefixed with __, regarding the status for the action on each record, as illustrated in the previous examples. The JSON success responses use the following syntax:

status

```
"__status": "<value>"
```

where <value> is success or failure.

JSON Failure Response

When the `_status` element returns failure, the `_error` element is added to identify the error and reason.

```
"__error": { "message": "<error value>", "reason": "<reason value> "}
```

where `<error value>` is the error message text and `<reason value>` is the reason the error was triggered.

The JSON error response contains only the error and reason elements. Generally, this indicates that the whole JSON operation failed and no records can be processed.

For example:

```
{"__error":"Cannot update with empty sysparm_query","reason":":null"}
```

Setting the number of rows returned

The following system property controls how many rows JSON returns with each query.

Setting the Number of Rows Returned

Property	Description
glide.processor.json.row_limit	<p>Specify the maximum number of rows a JSON query returns.</p> <ul style="list-style-type: none">Type: integerDefault value: 10,000

Property	Description
	<ul style="list-style-type: none">• Location:Add to the System Properties [sys_properties] table

Requiring basic authentication for incoming JSONv2 requests

The following system property controls whether basic authentication is required for incoming JSONv2 requests.

Requiring Basic Authentication for Incoming JSONv2 Requests

Property	Description
glide.basicauth.required.jsonv2	<p>Enables (true) or disables (false) requiring basic authentication for incoming JSONv2 requests.</p> <ul style="list-style-type: none">• Type: true false• Default value: true• Location:Add to the System Properties [sys_properties] table

Note: To learn more about this property, see [Basic auth: JSONv2 requests](#) in Instance Security Hardening Settings.

Action parameters

Action parameters are separate and different from data parameters because they specify the action to take when the JSON object parameter is part of an HTTP GET or POST request.

The parameters can also be specified as a field in the supplied JSON object. They have the effect of triggering an action in the case of

`sysparm_action`, or filtering the results of an update or query in the case of `sysparm_query`.

sysparm_action

The following are the valid values for `sysparm_action` and the corresponding action triggered by the API.

Data Retrieval

Method Summary	Description
getKeys	Query the targeted table using an encoded query string and return a comma delimited list of <code>sys_id</code> values.
getRecords	Query the targeted table using an encoded query string and return all matching records and their fields.
get	Query a single record from the targeted table by specifying the <code>sys_id</code> in the <code>sysparm_sys_id</code> URL parameter, and return the record and its fields.

Data Modification

Method Summary	Description
insert	Create one or more new records for the table targeted in the URL.
insertMultiple	Create multiple new records for the table targeted in the URL.
update	Update existing records in the targeted table in the URL, filtered by an encoded query string.
deleteRecord	Delete a record from the table targeted in the URL by specifying

Method Summary	Description
	its <code>sys_id</code> in the <code>sysparm_sys_id</code> URL parameter.
<code>deleteMultiple</code>	Delete multiple records from the table targeted in the URL, filtered by an encoded query string.

sysparm_query

Specify an encoded query string to be used in get, getRecords, update or deleteMultiple sysparm_action value.

sysparm_view

Specify a form view to customize the return values for get and getRecords function calls. When using a view, the query returns only the fields defined in the view, including referenced values. If there is no view name, or if the view name is not valid, then the query returns all field names that are marked active in the dictionary.

sysparm_sys_id

Specify a target sys_id during a get or delete function call (sysparm_action value).

sysparm_record_count

Specify an integer value to limit the number of records retrieved for this request. Note that this value is capped by the glide.processor.json.row_limit system property.

displayvalue

Get the display value of a reference field, if any are in the record. For example, the Incident record can have an assigned_to field that is a reference to a user record. Instead of sending the sys_id of the user record, the user name is sent.

The displayvalue parameter can have three values: **true**, **false**, or **all**.

- **true:** All the references fields show the display value instead of sys_id.

- **false** (default): All reference fields show `sys_ids`.
- **all**: The display value and the `sys_id` are shown. For example, the `assignedto` field in the Incident record is sent back as `assigned_to:1234556, dv_assigned_to:Fred Luddy`.

displayvariables

Set this boolean value to **true** during a get or getRecords function call to retrieve all variables attached to this record.

JSON Data Retrieval API

Query for data by issuing an HTTPS GET request to the instance.

By default, a GET request is interpreted as a get function if a `sysparm_sys_id` parameter is present. Otherwise, it is interpreted as a `getRecords` function. You can also specify a URL parameter `sysparm_action=get`. Query responses are always encapsulated by a records hash of records, where each individual record's values are themselves hashed by field name.

Return Display Value for Reference Variables

When you are getting a record from a get or `getRecords` function, all the fields associated with that record are returned. The fields are often reference fields that contain a `sys_id` for another table. The base system behavior is to return the `sys_id` value for those fields. To have the display value for the field returned, use one of these options:

- Add the property `glide.json.return_displayValue` to the system properties, and every JSON request will return a display value for a reference field.
- Add the parameter `displayvalue=true` to the JSON request URL and JSON requests with that parameter will return a display value instead of the `sys_id` for a reference field. The JSON URL would look like this:

```
https://<instance_name>.service-now.com/incident.do?JSON&  
sysparm_action=getRecords&sysparm_query=active=true^category=hardware&displayvalue=true
```

- Add the parameter `displayvalue=all` to the JSON request URL and JSON requests with that parameter return a display value and the `sys_id` for

a reference field. The response element name for the display value field will be prefixed with dv , for example dv caller id.

Get variables

Use the `displayvariables` query parameter to return an array of variables associated with a Service Catalog item record. To get variables, add the parameter `displayvariables=true` to the JSON request URL. For example, here is a URL to retrieve a record in JSON format that includes Service Catalog variables:

```
https://<your-instance>.servicenow.com/sc_req_item.do?JSON  
v2&sysparm_action=getRecords&sysparm_query= sys_id=5018da81  
742bd410f8771974894916fe&displayvariables=true
```

Here is the example response that displays a multi-row variable set from the record:

```
{
  "records": [
    {
      ...
      "variables": [
        {
          "display_value": [
            {
              "quantity": "1",
              "color": "Black",
              "device_type": "Apple iPhone 8",
              "storage": "64GB"
            },
            {
              "quantity": "1",
              "color": "Black",
              "device_type": "Apple iPhone 8",
              "storage": "64GB"
            }
          ],
          "columns_meta": [
            {
              "name": "device_type",
              "label": "Device Type",
              "id": "da7d3f3241411300964ff05369414ec
a",
              "type": 5,
            }
          ]
        }
      ]
    }
  ]
}
```

```
        "order":"0"
    },
{
    "name":"storage",
    "label":"Storage",
    "id":"691e337241411300964ff05369414e3
1",
    "type":5,
    "order":"1"
},
{
    "name":"color",
    "label":"Color",
    "id":"e89fb77241411300964ff05369414e7
4",
    "type":5,
    "order":"2"
},
{
    "name":"quantity",
    "label":"Quantity",
    "id":"2d5f737241411300964ff05369414ea
f",
    "type":5,
    "order":"3"
}
],
"max_rows":50,
"name":"mobile_devices_set",
"id":"e84d3f3241411300964ff05369414e3e",
"type":"one_to_many",
"value": [
{
    "quantity":"1",
    "color":"black",
    "device_type":"iphone8",
    "storage":"64GB"
},
{
    "quantity":"1",
    "color":"black",
    "device_type":"iphone8",
    "storage":"64GB"
}
]
```

```
        ],
        "row_count":2
    },
    {
        "question_text":"Department",
        "name":"department",
        "type":8,
        "value":"Development",
        "order":100
    },
    {
        "question_text":"Who is this request for?",
        "name":"requested_for",
        "type":8,
        "value":"System Administrator",
        "order":100
    },
    {
        "question_text":"When do you need this?",
        "name":"needed_by",
        "type":5,
        "value":"Today",
        "order":200
    },
    {
        "question_text":"Business Justification",
        "name":"business_justification",
        "type":2,
        "value":"Example justification",
        "order":200
    }
],
...
}
]
```

The keys in the response are defined as follows:

Key	Description
display_value	Multi-row variable set question display value. Only returned with multi-row variable sets.
columns_meta	Array of multi-row variable set metadata, such as the sys_id and name of the field. Only returned when the variable contains multiple fields.
max_rows	Maximum rows allowed in the multi-row variable set. Only returned with multi-row variable sets.
name	Question name.
id	Sys_id of the multi-row variable set. Only returned with multi-row variable sets.
type	Type of question.
value	Question value.
row_count	Current number of rows in the multi-row variable set. Only returned with multi-row variable sets.
question_text	Question label. Only returned with single-row variable sets.
order	Order of the question.

Control the order of records

You can control the order that records appear in the JSON response. To set an order, use the **ORDERBY** or **ORDERBYDESC** clauses in the URL encoded query. For example,

```
sysparm_query=active=true^ORDERBYnumber^ORDERBYDESCcategory
```

filters all active records and orders the results in ascending order by number first, and then in descending order by category. For more information, see [Encoded query strings](#).

getKeys

Get the `sys_id` of multiple records by specifying an encoded query string in the `sysparm_query` parameter.

```
https://<instance name>.service-now.com/incident.do?JSONv2  
&sysparm_action=getKeys&sysparm_query=active=true^category  
=hardware
```

get

Get a record directly by specifying the `sys_id` in a `sysparm_sys_id` parameter.

```
https://<instance name>.service-now.com/incident.do?JSONv2  
&sysparm_sys_id=9d385017c611228701d22104cc95c371
```

Optionally, you may also specify the `sysparm_action` parameter:

```
https://<instance name>.service-now.com/incident.do?JSONv2  
&sysparm_action=get&sysparm_sys_id=9d385017c611228701d2210  
4cc95c371
```

getRecords

Get all records by specifying an encoded query string in the `sysparm_query` parameter.

```
https://<instance name>.service-now.com/incident.do?JSONv2  
&sysparm_action=getRecords&sysparm_query=active=true^cate  
gory=hardware
```

JSON Data Modification API

Modify data using the JSON web service by sending an HTTPS POST request to the instance.

The HTTP POST must contain a `sysparm_action` parameter to indicate the type of action to be performed, with the incoming JSON object post in the body.

Note: The content-type of the POST should be `application/json`. It cannot be `application/x-www-form-urlencoded` or `multipart/form-data`.

insert

Create a new record in ServiceNow. The JSON object has to be POSTed as the body (content-type is usually application/json, although not enforced). The response from the record creation is a JSON object of the incident that was created.

For example, posting the following JSON object:

```
{"short_description": "this is a test", "priority": "1"}
```

to the following URL:

```
https://your_instance.service-now.com/incident.do?JSONv2&sysparm_action=insert
```

creates an incident.

Optionally, you may also specify the sysparm_action in the JSON object. The parameter inside the JSON object takes precedence over the URL parameter. For example:

```
{"sysparm_action": "insert", "short_description": "this is a test", "priority": "1"}
```

insertMultiple

To create multiple new records in ServiceNow, the input JSON object for the insert function must be an array. The response from the record creation is a JSON object of the incidents that were created. For example, the following JSON object:

```
{ "records" : [ { "short_description" : "this was inserted with python using JSON 1" , "priority" : "1 - Critical" , "impact" : "1" , "caller_id" : "Fred Luddy" } , { "short_description" : "this was inserted with python using JSON 2" , "priority" : "1 - Critical" , "impact" : "1" , "caller_id" : "Fred Luddy" } ] }
```

posted to one the following URLs:

```
https://<instance name>.service-now.com/incident.do?JSONv2&sysparm_action=insert  
https://<instance name>.service-now.com/incident.do?JSONv2&sysparm_action=insertMultiple
```

creates two incidents. Note the fields described as an array value for the **records** field.

update

Update a record or a list of records filtered by an encoded query string specified by the `sysparm_query` URL parameter. The JSON object has to be posted as the body (content-type is usually `application/json`, although not enforced). The response from the record creation is an array of JSON objects representing the records that were updated.

For example, posting the following JSON object:

```
{"short_description": "this was updated with python", "priority": "3", "impact": "1"}
```

to the following URL:

```
https://instance_name.service-now.com/incident.do?JSONv2&sysparm_query=priority=3&sysparm_action=update
```

updates all incidents with priority 3, and sets the values specified by the JSON object.

deleteRecord

Delete a single record from the targeted table, identified by a `sysparm_sys_id` parameter. The parameter may be encoded in the input JSON object or given as a URL parameter.

For example, posting:

```
{"sysparm_sys_id": "fd4001f80a0a0b380032ffa2b749927b"}
```

to the following URL:

```
http://instance_name.service-now.com/incident.do?JSONv2&sysparm_action=deleteRecord
```

deletes the incident record identified by the `sys_id` `fd4001f80a0a0b380032ffa2b749927b`.

deleteMultiple

Delete multiple records from the targeted table, filtered by an encoded query string specified in the `sysparm_query` URL parameter. The filter may also be encoded in the input JSON object.

For example, posting:

```
{"sysparm_query":"short_description=this was updated with  
python"}
```

to the following URL:

```
http://instance_name.service-now.com/incident.do?JSONv2&sy  
sparm_action=deleteMultiple
```

deletes all incident records where the `short_description` field contains the value "this was updated with python".

RSS web service

RSS (Rich Site Summary) is a format for delivering web-based information that changes regularly.

- [RSS feed generator](#)

ServiceNow supports the dynamic generation of RSS feeds.

- [RSS feed reader](#)

Create a scrolling RSS feed reader using a UI page.

RSS feed generator

ServiceNow supports the dynamic generation of RSS feeds.

Much like our Web Services implementation, the retrieval of an RSS feed representation of information is simply done by specifying an RSS parameter at the end of the URL to a table list. For example, the following will return a list of all incidents in RSS 2.0 format.:

Adding a Query

To associate a query to the list so that a filtered list is returned, use the `sysparm_query` parameter. For example, the following will return a list of all incidents where the priority field is 1 (Critical):

```
https://<instance_name>.service-now.com/incident.do?syspa  
rm_query=priority=1&RSS
```

If you have a multi part query then you would separate the parts with the ^ character. For example to get all priority 1 incidents with a category of software you would:

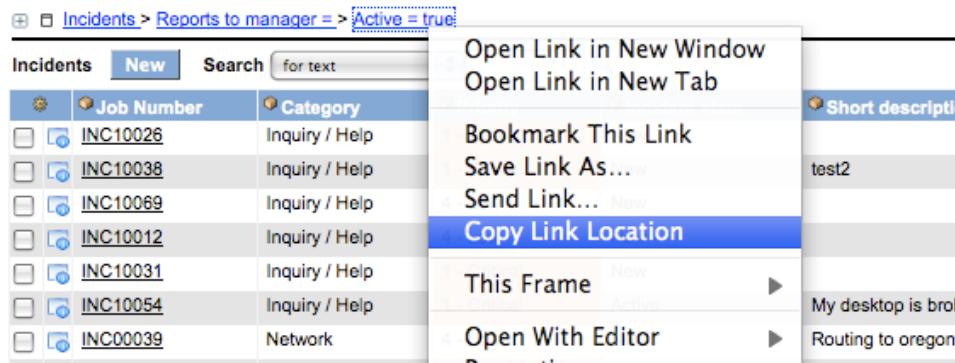
```
https://<instance name>.service-now.com/incident.do?sysparm_query=priority=1^category=software&RSS
```

If you want to query on a field that is a reference to another file then you need to use javascript to resolve the reference to the other file. For example, the assigned_to field in incident is a reference to a user record. If you wanted to find all the incidents assigned to "ITIL User" then you would do the following:

```
https://<instance name>.service-now.com/incident.do?sysparm_query=assigned_to=javascript:GetIDValue('sys_user', 'ITIL%20User')&RSS
```

Note: You can in most cases simply append "&RSS" to a URL that you generate in the UI or that of your favorite module. The easiest way to get the URL is to simply click the last breadcrumb from the list view. After appending "&RSS" then you can use this URL in your RSS feed reader

Get Bread URL



- Limiting results with a view

The description element in the returned RSS xml is constructed using the view as specified in the URL, when no view is specified, the default no-name view is used.

- Formatting results

The description element in the returned RSS xml can be formatted by setting the URL parameter sysparm_format=true and specifying the format string in the property glide.rss.description_format.

- [RSS basic authentication](#)

To enforce basic authentication on each request for an RSS feed, set the property glide.basicauth.required.rss to true.

- [RSS title override](#)

You may optionally override the automatically generated title of the RSS feed by added the sysparm_title parameter to the request URL.

The description element in the returned RSS xml is constructed using the view as specified in the URL, when no view is specified, the default no-name view is used.

To change this format, specify the sysparm_view parameter on the URL. For example, the following request will return the incidents list. However the result will be restricted to only the fields available in the ess view:

```
https://<instance name>.service-now.com/incident.do?sysparm_query=priority=1&sysparm_view=ess&RSS
```

Additionally, the RSS item title can be modified using the sysparm_title_view URL parameter. When specified, the item title will be constructed using the fields specified in the view. For example:

```
https://<instance name>.service-now.com/incident.do?sysparm_query=priority=1&sysparm_view=ess&sysparm_title_view=rss_title&RSS
```

The description element in the returned RSS xml can be formatted by setting the URL parameter sysparm_format=true and specifying the format string in the property glide.rss.description_format.

By default, when the URL parameter is present, the description element will be formatted to contain the field label and value using the following format string:

```
<b>{1}</b>: {2}<br/>
```

- {0} - field name
- {1} - field label

- {2} - field value

This default format string can be overridden using the property glide.rss.description_format. An example of the formatted RSS feed can be seen in the following screen capture from Firefox:

RSS Format

Problem (All)

Problem (All)

PRB00001

Sun, Aug 10, 2008 3:39 PM

Number: PRB00001

Escalation: Moderate

Short description: Windows xp SP2 causing errors in Enterprise

Problem state: Pending Change

RFC: CHG00003

PRB00008

Sun, Aug 10, 2008 3:40 PM

Number: PRB00008

Escalation: Normal

Short description: Hang when trying to print VISIO document

Problem state: Open

RFC:

To enforce basic authentication on each request for an RSS feed, set the property glide.basicauth.required.rss to true.

RSS request would have to contain the Authorization header as specified in the Authentication protocol. Because the request is non-interactive, we always require the Authorization header during a request.

Note: If you plan to disable RSS basic authentication, make sure that tables in the platform have the right ACL entries to protect from unauthorized access.

To specify basic authentication on the URL, put the username and password pair separated by a colon in front of the server name before an @ character. For example, to submit the demo credentials for the ITIL user, use the following URL.

```
https://itil:itil@<instance name>.service-now.com/incident  
.do?RSS
```

Some older browsers, such as Microsoft IE 7 do not support direct URL authentication. If the site uses basic authentication, Internet Explorer automatically prompts users for a user name and a password. In some cases, users can click the Remember my password box in the prompt to save their credentials for later visits to that site.

You may optionally override the automatically generated title of the RSS feed by added the sysparm_title parameter to the request URL.

For example, you can specify the title Priority One Incidents using the following request URL.

```
https://<instance name>.service-now.com/incident.do?sysparm_query=priority=1&sysparm_view=ess&RSS&sysparm_title=Priority%20One%20Incidents
```

This will produce results as follows:

RSS Out

```
<?xml version="1.0" encoding="UTF-8" ?>
- <rss version="2.0">
- <channel>
  <title>Priority One Incidents</title>
  <link>http://www.service-now.com/demo/nav_to.do?uri=incident.do?sysparm_query=priority=1</link>
  <description>Priority One Incidents</description>
  <copyright>Copyright 2006 Service-now.com</copyright>
  <pubDate>Mon, 12 Jun 2006 11:04:44 PDT</pubDate>
  <lastBuildDate>Mon, 12 Jun 2006 11:04:44 PDT</lastBuildDate>
  <generator>jRSSGenerator by Henrique A. Viecili</generator>
  <docs>http://blogs.law.harvard.edu/tech/rss</docs>
- <item>
  <title>INC00009</title>
  <link>http://www.service-now.com/demo/nav_to.do?uri=incident.do?
    sys_id=46b66a40a9fe198101f243dfbc79033d%26sysparm_stack=incident_list.do%0
    3Fsysparm_query=active=true</link>
  <description>INC00009 2006-02-01 14:50:23 Reset my password</description>
  <author>glide.main</author>
  <guid>46b66a40a9fe198101f243dfbc79033d</guid>
  <pubDate>Wed, 17 May 2006 18:20:57 PDT</pubDate>
</item>
+ <item>
+ <item>
+ <item>
+ <item>
+ <item>
</channel>
</rss>
```



RSS feed reader

Create a scrolling RSS feed reader using a UI page.

You must create a feed parser using an RSS API, such as the Google Feed API.

Developer's guide: <http://code.google.com/apis/ajaxfeeds/documentation/>

API: <http://code.google.com/apis/ajaxfeeds/documentation/reference.html>

- [Scrollable areas](#)

A scrollable area is a div where contents scroll from the bottom up over time.

- [Add a scrolling news panel to your homepage](#)

Display knowledge article content in a scrolling news panel.

- [News in the scroller](#)

The news scroller gets its news list by going to the Knowledge Base and querying the short descriptions of any items with a topic of "News".

- [RSS feed reader example](#)

An example of how to set up an RSS feed reader using an RSS feed.

A scrollable area is a div where contents scroll from the bottom up over time.

You can scroll any HTML content, and anything inside the scroller is operational HTML with functioning links and images.

To make a scrollable areas, wrap the scrolling content inside of a scrollable_area tag, likely in a UI Page:

```
<g:scrollable_area height="100px">
    <g2:evaluate var="jvar_temp" expression="var kb =
new GlideRecord('kb_knowledge');"/>
    <g:inline template="kb_section.xml"/>
</g:scrollable_area>
```

The system will then create a 100 pixel high div and the contents will automatically scroll from bottom to top. If you have a 1000 pixel high block of text, for example, you'll see the top 100 pixels and then pixels 2-101, then 3-102, etc. Once it reaches the top it'll wrap back around to the bottom.

This sample code will create a scroller with a list of priority 1 incidents.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide"
  xmlns:j2="null" xmlns:g2="null">
  <g2:evaluate var="jvar_inc">
    var inc = new GlideRecord('incident');
    inc.addActiveQuery();
    inc.addQuery('priority',1);
    inc.query();
  </g2:evaluate>

  <g2:scrollable_area height="100px">
    <table border="0" cellspacing="2" cellpadding="0" width="100%">
      <j2:while test="${inc.next()}"/>
        <j2:set var="jvar_inc_link" value="incident.do?sys_id=${inc.sys_id}"/>
        <j2:set var="jvar_inc_list_link" value="incident_list.do?sysparm_query=active=true"/>
        <tr>
          <td>
            <a href="${jvar_inc_link}">
              <IMG SRC="images/services.png" style="padding-right:10px"></IMG>
            </a>
            <a href="${jvar_inc_link}" style="padding-right:10px; color:blue">${inc.number}</a>
          </td>
          <td>${inc.short_description}</td>
        </tr>
      </j2:while>
      <tr>
        <td align="center" colspan="2"><a href="${jvar_inc_list_link}" style="color:blue">View all active incidents</a></td>
      </tr>
    </table>
  </g2:scrollable_area>
</j:jelly>
```

- Add scrolling elements in forms

You can add scrolling areas to forms as well as UI pages.

You can add scrolling areas to forms as well as UI pages.

1. Create a UI Macro with the script.
 2. Create a Formatter to reference the script.
- [Priority 1 incidents example](#)

This example scrolling element demonstrates how to create a UI macro to a scrolling list of priority 1 incidents.

This example scrolling element demonstrates how to create a UI macro to a scrolling list of priority 1 incidents.

Use the following example code:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide"
" xmlns:j2="null" xmlns:g2="null">
    <g2:evaluate var="jvar_inc">
        var inc = new GlideRecord('incident');
        inc.addActiveQuery();
        inc.addQuery('priority',1);
        inc.query();
    </g2:evaluate>

    <div style="background-color:DDDDDD; padding-left:10px;
line-height:19px; border:2px white solid" width="100%" nowrap="true">
        Priority 1 Incidents:
        <input id="make_spacing_ok" style="visibility:hidden; width:0px;" title="" />
    </div>
    <g2:scrollable_area height="100px" width="100%">

        <j2:while test="$[inc.next()]">
            <j2:set var="jvar_inc_link" value="incident.do?sys
_id=$[inc.sys_id]" />
            <j2:set var="jvar_inc_list_link" value="incident_l
ist.do?sysparm_query=active=true" />
            <span style="line-height: 10px; padding-left:10px">
        <a href="$[jvar_inc_link]">
```

```
<IMG SRC="images/services.png" style="padding-right:10px"></IMG>
</a>
<a href="${jvar_inc_link}" style="padding-right:10px; color:blue">${inc.number}</a>
    ${inc.short_description}
</span>
<br style="line-height:5px"/>
</j2:while>
<span>
    <a href="${jvar_inc_list_link}" style="color:blue; padding-left:10px">View all active incidents</a>
</span>

</g2:scrollable_area>
</j:jelly>
```

Navigate to **System UI > Formatters** and create a Formatter that refers to the UI Macro above.

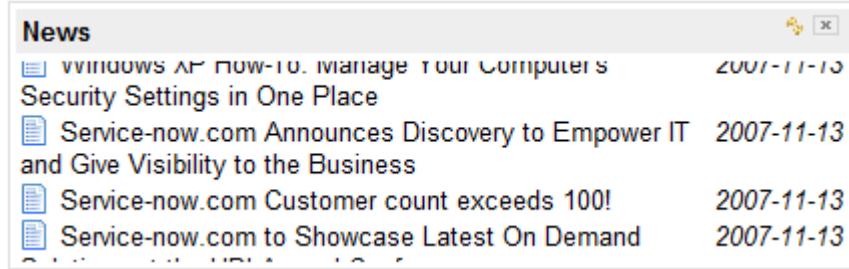
Display knowledge article content in a scrolling news panel.

Before you begin

- Role required: none
- Create knowledge content to display in the news panel.

Procedure

1. Navigate to a dashboard.
2. Click **Add Content** to get a list of possible content types.
3. Click **Gadgets** in the leftmost box of the add content dialog.
4. Click the news scroller.
5. Click **Add** to add it on your page.



The news scroller gets its news list by going to the Knowledge Base and querying the short descriptions of any items with a topic of "News".

To add news, go to the Knowledge Base and add new entries with a topic of "News".

- [Scrolling News Panel Notes and Limitations](#)

Scollable news panels offer a way for you to display high impact news items on your users home pages in an eye catching scrolling news display.

Scollable news panels offer a way for you to display high impact news items on your users home pages in an eye catching scrolling news display.

- Scrolling is currently always from the bottom to the top.
- Scrolling is 1 pixel a at a time every 100 ms.
- You can have as many scrollable areas on a screen as you want (they get system generated unique names).
- Works in Internet Explorer and Firefox.
- If the object you are scrolling is shorter than the scrollable area, it still scrolls.
- If you mouse over a scrollable area, it stops scrolling (so you can click something without chasing it as it scrolls).

An example of how to set up an RSS feed reader using an RSS feed.

For an example RSS feed reader, go to the [ServiceNow Community](#) site and search for **RSS Feed Reader**.

Exporting and converting records into complex data types

Use URL parameters to export table records and convert them into complex data types, such as JSON, XML, PDF, CSV, and XLS.

Exporting records as complex data types

You can use an HTTP GET request to retrieve records from a table and put them in a specified format. For example, use the PDF parameter in a GET request to export records from a table as PDF files; use the XLS parameter to export records from a table as XLS files. For example, to retrieve a list of incident records as XLS files, issue an HTTP GET using the following URL: https://instance_name.service-now.com/incident.do?XLS. The file returned is incident.xls. incident.do is basically a GET that returns a list of the records from the incident table. The XLS parameter converts those records into XLS files.

The general syntax is: `https://<serviceNow-instance-name>/<table-name>.do?<Data-type-parameter>`

URL parameters

The following table shows URL parameters you can use in GET requests, filters you can use to filter out unwanted table records in the return, and an indicator of whether you can POST the data type directly to a table. The parameter becomes the extension of the returned file, for example, using the XLS parameter returns a file in the form <table-name>.xls.

URL parameters

Data type	Parameter	Valid filters	Directly POST to table?
CSV	CSV	sysparm_query, sysparm_view	Y
Excel	XLS, EXCEL, XLSX	sysparm_query, sysparm_view	Y
JSON	JSONv2	Various. See JSON data retrieval API .	Y

Data type	Parameter	Valid filters	Directly POST to table?
PDF	PDF	sysparm_query, sysparm_view	N
RSS	RSS	sysparm_query, sysparm_view and more. See Limiting results with a view .	N
XML	XML, XSD, SCHEMA	sysparm_query, useUnloadFormat	N

For more information about retrieving and converting table records into the JSON file format, see [JSONv2 Web Service](#).

For more information about retrieving and converting table records into the RSS file format, see [RSS feed generator](#).

Converting records to PDFs

For PDF export, there is a distinction between targeting a table and targeting its list. To generate a PDF of a list of records, suffix the target with _list. To target a single record, you must specify the sys_id parameter to identify the record for which you are generating the PDF.

Filters

All URL parameters work with filters that enable you to export a subset of table records. For example, sysparm_query=active=true in a GET request exports only active records. The following example exports only active incident records in an Excel format: https://instance_name.service-now.com/incident.do?EXCEL&sysparm_query=active=true.

The general syntax is: `https://<serviceNow-instance-name>/<table_list>.do?<Data-type-parameter>&<filter>`

Filters include:

- `sysparm_query`—Filters the data using the encoded query before exporting files, for example, `sysparm_query=active=true` exports only active records.
- `sysparm_view`—Specify the name of a list view to control which fields are returned. For example, to return the ESS view, use `sysparm_view=ess`.
- `useUnloadFormat`—Indicates that the XML format returned is an unload format. The unload format is the same format you get when, from a list in the UI, you select Export > XML > ... You can import unload-formatted XML files back into the tables. To enable the unload format from a URL, use the `useUnloadFormat=true` URL parameter, for example, `https://instance_name.service-now.com/incident.do?XML&useUnloadFormat=true`.

Example GET queries

GET request examples

Data type	Example query
CSV	<code>https://instance_name.service-now.com/incident.do?CSV&sysparm_query=active=true</code>
Excel	<code>https://instance_name.service-now.com/incident.do?XLS&sysparm_query=active=true</code>
PDF	<code>https://instance_name.service-now.com/incident.do?PDF&sysparm_view=ess</code>
RSS	<code>https://instance_name.service-now.com/incident.do?RSS&sysparm_view=ess</code>
XML	<code>https://instance_name.service-now.com/incident.do?XML&sysparm_query=active=true</code>

Returned files

GET queries return records from a table in the format specified in the request. For example, a query that uses the XLS parameter returns a table record in a file with the .xls extension.

The Content-Disposition header in the response displays the file name and extension of the returned file. The file name is based on the table you export from, such as incident.xls, incident.pdf, or incident.xml.

Exporting data into tables

You can POST the following data types directly into tables:

- CSV
- Excel
- JSON

The file headers must match the field columns in the targeted table. For more information, see [Post CSV or Excel files directly to an import set](#).

Inbound web service examples

Inbound web service examples demonstrate how to access ServiceNow web services.

- [Java Apache Axis2 web services client examples](#)

Examples demonstrating an integration with Axis2 Version 1.4.

- [Microsoft .NET web services client examples](#)

Examples demonstrating an integration with Microsoft .NET Web Services Client.

- [Perl web services client examples](#)

Examples demonstrating an integration with a Perl web services client.

- [Python web services client examples](#)

Examples demonstrating an integration with a Python web services client.

- [Web services C Sharp .NET end to end tutorial](#)

Examples demonstrating how to use .NET to consume a ServiceNow web service.

- Retrieve a large number of records using SOAP

By default, a single SOAP request can retrieve a maximum of 250 records.

Java Apache Axis2 web services client examples

Examples demonstrating an integration with Axis2 Version 1.4.

Requirements

- An "elementFormDefault" value of qualified means that an unqualified element is in the default namespace defined on an ancestor. If it is "unqualified" then an unqualified element is in the empty namespace (xmlNs=""). The default is "unqualified".
- To Resolve the Axis Client deserialization failure you should go to **System Properties > Web Services** and uncheck the property that sets the elementFormDefault attribute of the embedded XML schema to the value of unqualified. Save the property setting and regenerate your Axis2 client code if your client code was generated before changing this property.

Element Form Default Property

This property sets the elementFormDefault attribute of the embedded XML schema to the value of unqualified, if set to true. This attribute indicates whether or not locally declared elements must be qualified by the target namespace in an instance document. If the value of this attribute is 'unqualified', then locally declared elements should not be qualified by the target namespace. If the value of this attribute is 'qualified', then locally declared elements must be qualified by the target namespace. For compatibility with Clients generated from WSDL (.NET Web Reference, Axis2 stub, webMethods, etc.), set this value to false. This value defaults to true.

For further documentation, follow this URL http://wiki.service-now.com/index.php?title=Web_Services

Yes | No

- Java Apache Axis2 web services client examples insert

An example class to insert an incident record.

- Java Apache Axis2 web services client examples update

An example of an Axis Client program that calls the getKeys function to query all incidents where the category is Hardware.

- Java Apache Axis2 web services client examples advanced

Examples showing how to construct and use an Axis2 client to consume a ServiceNow Web Service.

An example class to insert an incident record.

```
public class Insert {  
  
    public static void main ( String args [ ] ) { try {  
        HttpTransportProperties. Authenticator basicAuthenticat  
ion = new HttpTransportProperties. Authenticator ( ) ;  
        basicAuthentication. setUsername ( "admin" ) ;  
        basicAuthentication. setPassword ( "admin" ) ;  
  
        ServiceNowStub proxy = new ServiceNowStub ( ) ;  
  
        proxy._getServiceClient ( ). getOptions ( ). setPropert  
y (org. apache. axis2. transport. http. HTTPConstants. CHU  
NKED, Boolean. FALSE ) ;  
        proxy._getServiceClient ( ). getOptions ( ). setPropert  
y (org. apache. axis2. transport. http. HTTPConstants. AUT  
HENTICATE, basicAuthentication ) ;  
  
        ServiceNowStub. Insert inc = new ServiceNowStub. Inser  
t ( ) ;  
        ServiceNowStub. InsertResponse resp = new ServiceNowSt  
ub. InsertResponse ( ) ;  
  
        inc. setAssigned_to ( "Christen Mitchell" ) ;  
        inc. setCategory ( "hardware" ) ;  
        inc. setPriority ( BigInteger. ONE ) ;  
        inc. setDescription ( "The WI_FI in the reception area  
is down" ) ;  
        inc. setCaller_id ( "Joe Employee" ) ;  
  
        resp = proxy. insert (inc) ;  
  
        System. out. println ( "New Incident: " + resp. getNum  
ber ( ) ) ; } catch ( Exception e ) { System. out. printl  
n (e. toString ( ) ) ; }  
    } }
```

An example of an Axis Client program that calls the `getKeys` function to query all incidents where the category is Hardware.

getKeys

A list of `sys_id` is returned as a result:

```
package com.service_now.www ;

public class DemoClient {

    public static void main ( String args [ ] ) { try {
        ServiceNowStub proxy = new ServiceNowStub ( ) ;
        ServiceNowStub. GetKeys getInc = new ServiceNowStub. G
etKeys ( ) ;
        ServiceNowStub. GetKeysResponse resp = new ServiceNowS
tub. GetKeysResponse ( ) ;

        getInc. setActive ( true ) ;
        getInc. setCategory ( "hardware" ) ;

        proxy._getServiceClient ( ). getOptions ( ). setPropert
y (org. apache. axis2. transport. http. HTTPConstants. CHU
NKED, Boolean. FALSE ) ;

        resp = proxy. getKeys (getInc ) ;

        String [ ] keys = resp. getSys_id ( ) ;

        System. out. println ( "Key: " + keys [ 0 ] ) ; } catc
h ( Exception e ) { System. out. println (e. toString ( )
) ; }

    } }
```

getRecords

```
package com.service_now.www ;

import com.service_now.www.ServiceNowStub.GetRecordsResul
t_type0 ;

public class GetRecords {

    /**
     * @param args
     */
    public static void main ( String [ ] args ) { try {
        ServiceNowStub proxy = new ServiceNowStub ( ) ;
        ServiceNowStub. GetRecords incidents = new ServiceNowSt
ub. GetRecords ( ) ;
```

```
    ServiceNowStub. GetRecordsResponse result = new Service
NowStub. GetRecordsResponse ( ) ;

    incidents. setActive ( true ) ;
    incidents. setCategory ( "hardware" ) ;
    incidents. setSys_created_on ( "> 2009-06-08 10:30:00"
) ;

    proxy._getServiceClient ( ). getOptions ( ). setPropert
y (org. apache. axis2. transport. http. HTTPConstants. CHU
NKED, Boolean. FALSE ) ;

    result = proxy. getRecords (incidents ) ;

    GetRecordsResult_type0 [ ] keys = result. getGetRecords
Result ( ) ;

    for ( int key = 0 ; key < keys. length ; key ++ ) { S
ystem. out. println ( "Key: " + keys [ 0 ]. getSys_id ( )
) ; } } catch ( Exception e ) { System. out. println (e. t
oString ( ) ) ; }

}
```

```
}
```

Examples showing how to construct and use an Axis2 client to consume a ServiceNow Web Service.

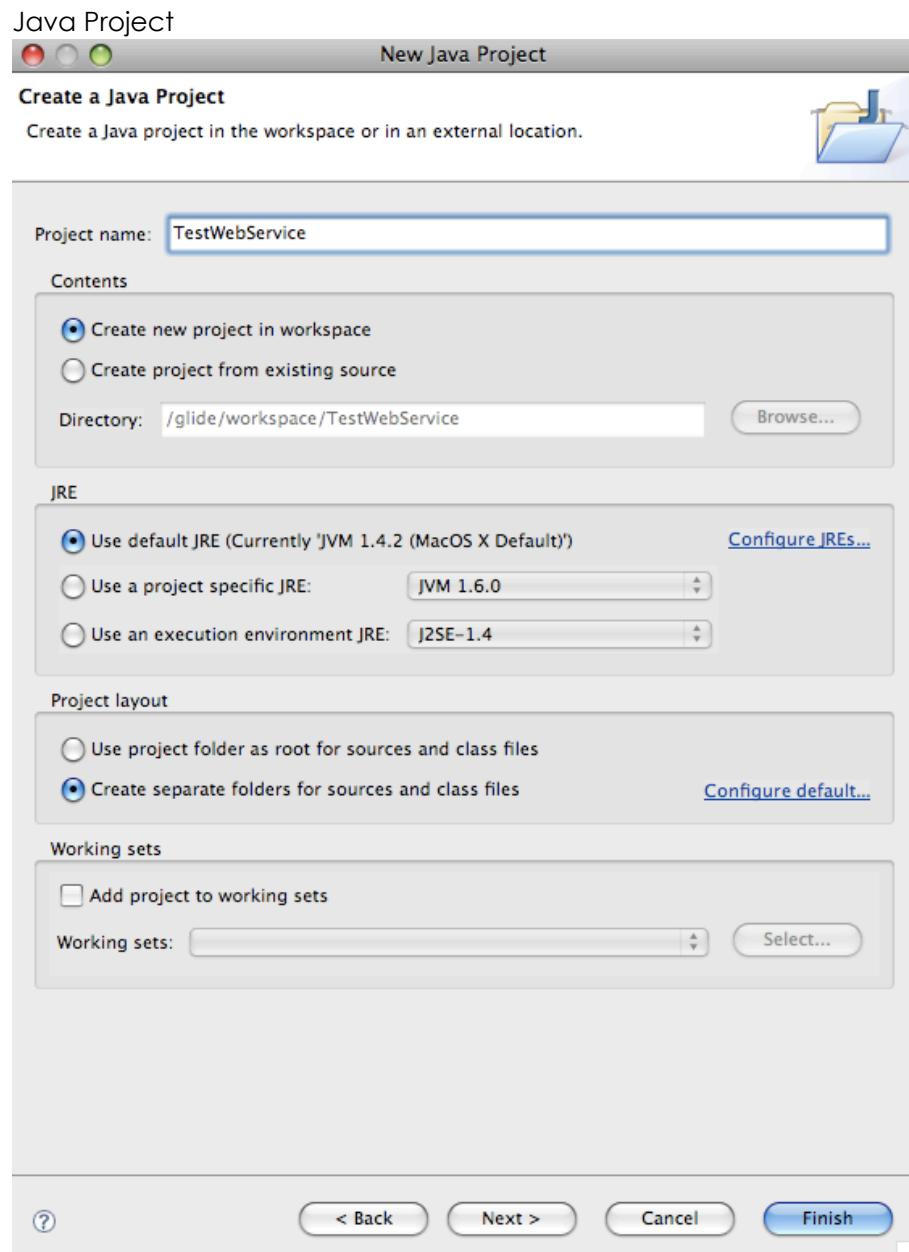
Axis is essentially a SOAP engine -- a framework for constructing SOAP processors such as clients, servers, or gateways. The current version of Axis is written in Java. This content is intended for system admins with a light development background in Java. To begin you would need Java JDK version 1.4.2 or higher and Axis2 version 1.0 or higher.

Create a Java Project

This example uses Eclipse SDK Version: 3.4.2 for managing the source code and executing the web request. Eclipse is not required.

- Open Eclipse and from the menu select **File > New > Project > Java Project**.

- Give the project a name.
- Verify that the correct JRE is specified.
 - If using wsdl2java run "java -version" on the command line and this will be the version to specify for the project specific JRE.
 - If using the Axis2 Codegen plugin use default JRE.



Generate your Axis2 client code

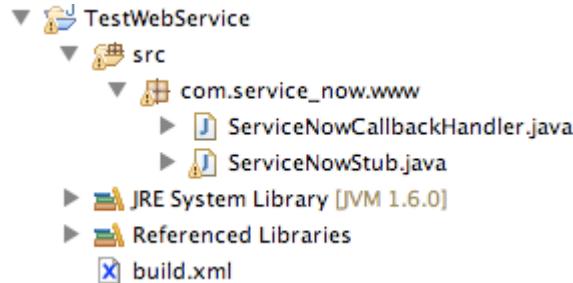
- From a command line in the bin directory of the axis folder:

```
./wsdl2java.sh -uri https://<instance name>.service-now.com/incident.do?WSDL -o /glide/workspace/TestWebService/
```

- In the above example:
 - The "-uri" is either the path where you have saved a copy of the wsdl to either ".wsdl" or ".xml", or the URL the WSDL resides at.
 - The "-o" is the path where you want the files to be written out to. If not specified, the files will be written out to the current bin location.

- In Eclipse refresh the project and the generated Stub and CallbackHandler should now be displayed

Axis Stub



Basic Authentication

```
HttpTransportProperties.Authenticator basicAuthenticatio
n = new HttpTransportProperties. Authenticator ( ) ;
basicAuthentication. setUsername ( "admin" ) ;
basicAuthentication. setPassword ( "admin" ) ;
...
ServiceNowStub proxy = new ServiceNowStub ( ) ;
...

proxy._getServiceClient ( ). getOptions ( ). setProperty
(org. apache. axis2. transport. http. HTTPConstants. AUTHE
NTICATE, basicAuthentication) ;
```

Compatibility with Axis2 Versions 1.1 and higher

Chunking support is only available in HTTP Version 1.1. By default chunking is enabled in Axis2.xml for versions 1.1 and higher. ServiceNow does not support Chunking, so you will need to disable chunking at deployment time or at runtime.

- Deployment time: One can disable HTTP chunking by removing or commenting out the following element from Axis2.xml

```
<parameter name= "Transfer-Encoding" >chunked</parameter>
```

- Runtime: User can disable the chunking using following property set in Client or Stub, versions 1.1.1 and higher only

```
options.setProperty (org. apache. axis2. transport. http  
. HTTPConstants. CHUNKED, Boolean. FALSE ) ;
```

Creating Unique Packages

You can use the Axis2 parameter namespace2package (ns2p) to create unique package names. The parameter uses this format:

```
<Axis path>\bin\wsdl2java.bat -u -p cr2 -ns2p <namespace>=  
<package name> -uri <wsdl to convert>
```

For example:

```
<Axis path>\bin\wsdl2java.bat -u -p cr2 -ns2p http://www.s  
ervice-now.com/change_request=my.change_request -uri chang  
e_request
```

Microsoft .NET web services client examples

Examples demonstrating an integration with Microsoft .NET Web Services Client.

Requirements

.NET 2.0 Versions and Higher:

- An "elementFormDefault" value of qualified means that an unqualified element is in the default namespace defined on an ancestor. If it is "unqualified" then an unqualified element is in the empty namespace (xmlns=""). The default is "unqualified".
- To Resolve the .NET Client deserialization failure you should go to **System Properties > Web Services** and uncheck the property that sets the elementFormDefault attribute of the embedded XML schema to the value of unqualified. Save the property setting and recreate your WSDL Reference.cs class. See Also "Compatibility with Clients generated from WSDL" below.

Element Form Default Property

This property sets the elementFormDefault attribute of the embedded XML schema to the value of unqualified, if set to true. This attribute indicates whether or not locally declared elements must be qualified by the target namespace in an instance document. If the value of this attribute is 'unqualified', then locally declared elements should not be qualified by the target namespace. If the value of this attribute is 'qualified', then locally declared elements must be qualified by the target namespace. For compatibility with Clients generated from WSDL (.NET Web Reference, Axis2 stub, webMethods, etc.), set this value to false. This value defaults to true.

For further documentation, follow this URL http://wiki.service-now.com/index.php?title=Web_Services

Yes | No

Perl web services client examples

Examples demonstrating an integration with a Perl web services client.

Note: The following examples require the usage of the [Perl language](#) and the [SOAP::Lite package](#).

System Requirements

Perl 5.8

- SOAP::Lite (prerequisites <http://soaplite.com/prereqs.html>)
- Crypt::SSLeay
- IO::Socket::SSL

insert

The following example will insert a record into the Incident table.

```
#!/usr/bin/perl -w

# declare usage of SOAP::Liteuse SOAP::Lite;

# specifying this subroutine, causes basic auth to use# it
# s credentials when challengedsub SOAP::Transport::HTTP::Client::get_basic_credentials{# login as the itil userreturn
'itil'=>'itil';}

# declare the SOAP endpoint heremy$soap= SOAP::Lite->proxy
('https://myinstance.service-now.com/incident.do?SOAP');

# calling the insert functionmy$method= SOAP::Data->name('
insert')->attr({xmlns =>'http://www.service-now.com/'});
```

```
# create a new incident with the following short_description and category
my@params=( SOAP::Data->name(short_description =>'This is an example short description'));push(@params, SOAP::Data->name(category =>'Hardware'));

# invoke the SOAP call
my$result=$soap->call($method=>@params);

# print any SOAP faults that get returned
print_fault($result);# print the SOAP response that get return
print_result($result);

# convenient subroutine for printing all resultssub print_result {my($result)=@_;

    if($result->body&&$result->body->{'insertResponse'}) {my%keyHash=%{$result->body->{'insertResponse'}};foreach my$k(keys%keyHash) {print"name=$k      value=$keyHash{$k}\n";}}
```

```
# convenient subroutine for printing all SOAP faultssub print_fault {my($result)=@_;

    if($result->fault) {print"faultcode=".$result->fault->{'faultcode'}."\n";print"faultstring=".$result->fault->{'faultstring'}."\n";print"detail=".$result->fault->{'detail'}."\n";}}
```

insert (With XML payload)

The following is an example of inserting a record into the `ecc_queue` table where the payload field is an XML document. This is done using the [Perl language](#) and the [SOAP::Lite](#) package, the XML document creation uses the [XML::Writer package](#):

```
#!/usr/bin/perl -w
use SOAP::Lite ( +trace => all, maptype => {} );
use SOAP::Lite;
use XML::Writer;
use XML::Writer::String;

## Get parameters passed by OVO notification
$OVMMSG{id}=$ARGV[0];
$OVMMSG{node_name}=$ARGV[1];
$OVMMSG{node_type}=$ARGV[2];
$OVMMSG{date_created}=$ARGV[3];
$OVMMSG{time_created}=$ARGV[4];
$OVMMSG{date_received}=$ARGV[5];
$OVMMSG{time_received}=$ARGV[6];
$OVMMSG{application}=$ARGV[7];
$OVMMSG{msg_group}=$
```

```
ARGV[8];$OVMSG{object}=$ARGV[9];$OVMSG{severity}=$ARGV[10];
:$OVMSG{operator_list}=$ARGV[11];$OVMSG{msg_text}=$ARGV[12];
:$OVMSG{instruction}=$ARGV[13];

sub SOAP::Transport::HTTP::Client::get_basic_credentials{return'itil'=>'itil';}

my$soap= SOAP::Lite->proxy('http://<instance name>.service
-now.com/ecc_queue.do?SOAP');

my$method= SOAP::Data->name('insert')->attr({xmlns =>'http
://www.service-now.com/'});

# get all incidents with category Networkmy@params=( SOAP:
:Data->name(agent =>'OVO_Notification'));push(@params, SOA
P::Data->name(queue =>'input'));push(@params, SOAP::Data->
name(name =>'HP Openview OVO Notification'));push(@params
, SOAP::Data->name(source =>$OVMSG{id}));

my$s= XML::Writer::String->new();my$writer=new XML::Writer
(OUTPUT =>$s);

#$writer->xmlDecl();$writer->startTag('notification');

write_element('id');
write_element('node_name');
write_element('node_type');
write_element('date_created');
write_element('time_created');
write_element('date_received');
write_element('time_received');
write_element('application');
write_element('msg_group');
write_element('object');
write_element('severity');
write_element('operator_list');
write_element('msg_text');
write_element('instruction');

$writer->endTag('notification');

$writer->end;

sub write_element {my$label=shift;my$value=$OVMSG{$label};}
```

```
$writer->startTag($label);if($value){$writer->characters($value);} $writer->endTag($label);}

push(@params, SOAP::Data->name(payload =>$s->value()));

print$soap->call($method=>@params)->result;</pre>

==== Response to the ''insert''====<source lang="xml"><?xml version="1.0" encoding="UTF-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soap:Body><insertResponse xmlns="http://www.service-now.com/ecc_queue"><sys_id>1a5ad50e0a0a021101bef2e07705f87a</sys_id><name>HP Openview OVO Notification</name></insertResponse></soap:Body></soap:Envelope>
```

update

```
#!/usr/bin/perl -w

use SOAP::Lite ( +trace => all, maptype => {} );use SOAP::Lite;

sub SOAP::Transport::HTTP::Client::get_basic_credentials{return'itil'=>'itil';}

my$soap= SOAP::Lite->proxy('http://localhost:8080/glide/incident.do?SOAP');

my$method= SOAP::Data->name('update')->attr({xmlns =>'http://www.service-now.com/'});

# update incident by sys_id my@params=( SOAP::Data->name(sys_id =>'e8caedcbc0a80164017df472f39eaed1'));push(@params,
SOAP::Data->name(short_description =>'this is a new description'));

my$result=$soap->call($method=>@params);

print_fault($result);
print_result($result);

sub print_result {my($result)=@_;
```

```
if($result->body&&$result->body->{ 'updateResponse' }) {my%  
keyHash=%{$result->body->{ 'updateResponse' }};foreach my$key(k  
eys%keyHash) {print"name=$k    value=$keyHash{$k}\n";}}}  
  
sub print_fault {my($result)=@_;  
  
    if($result->fault){print"faultcode=".$result->fault->{ 'f  
aultcode'}."\n";print"faultstring=".$result->fault->{ 'faul  
tstring'}."\n";print"detail=".$result->fault->{ 'detail'}."  
\n";}}
```

getKeys

The following is an example of retrieving a list of s for records of Incident where category is Network.

```
#!/usr/bin/perl -w  
  
use SOAP::Lite ( +trace => all, maptype => {} );use SOAP:  
:Lite;  
  
sub SOAP::Transport::HTTP::Client::get_basic_credentials{r  
eturn'itil'=>'itil';}  
  
my$soap= SOAP::Lite->proxy('http://<instance name>.service  
-now.com/incident.do?SOAP');  
  
my$method= SOAP::Data->name('getKeys')->attr({xmlns =>'htt  
p://www.service-now.com/'});  
  
# get all incidents with category Networkmy@params=( SOAP:  
:Data->name(category =>'Network'));  
  
print$soap->call($method=>@params)->result;
```

get

The following is an example of retrieving an Incident record using its sys_id value.

```
#!/usr/bin/perl -w#use SOAP::Lite ( +trace => all, maptyp  
e => {} );use SOAP::Lite;
```

```
sub SOAP::Transport::HTTP::Client::get_basic_credentials{r
eturn'itil'=>'itil';}

my$soap= SOAP::Lite->proxy('http://<instance name>.service
-now.com/incident.do?SOAP');

my$method= SOAP::Data->name('get')->attr({xmlns =>'http://
www.service-now.com/'});

# get incident by sys_idmy@params=( SOAP::Data->name(sys_i
d =>'9d385017c611228701d22104cc95c371'));

my%keyHash=%{$soap->call($method=>@params)->body->{ 'getResponse' }};

# iterate through all fields and print themforeachmy$k(key
s$keyHash){print"$k=$keyHash{$k}\n";}
```

getRecords

To query for an Incident using its incident number value:

```
#!/usr/bin/perl -w#use SOAP::Lite ( +trace => all, maptyp
e => {} );use SOAP::Lite;

sub SOAP::Transport::HTTP::Client::get_basic_credentials{r
eturn'itil'=>'itil';}

my$soap= SOAP::Lite->proxy('http://<instance name>.service
-now.com/incident.do?SOAP');

my$method= SOAP::Data->name('getRecords')->attr({xmlns =>''
http://www.service-now.com/'});# get incident by numbermy@
params=( SOAP::Data->name(number =>'INC10001'));

my%keyHash=%{$soap->call($method=>@params)->body->{ 'getRec
ordsResponse' }->{ 'getRecordsResult' }};

# iterate through all fields and print themforeachmy$k(key
s$keyHash){print"$k=$keyHash{$k}\n";}
```

getRecords (Returning Multiple Results)

The following is an example of retrieving and displaying an array of Incident records by querying all Incidents that have a of "Network"

```
#!/usr/bin/perl -w#use SOAP::Lite ( +trace => all, maptyp
e => {} );use SOAP::Lite;

sub SOAP::Transport::HTTP::Client::get_basic_credentials{r
eturn'itil'=>'itil';}

my$soap= SOAP::Lite->proxy('http://<instance name>.service
-now.com/incident.do?SOAP');

my$method= SOAP::Data->name('getRecords')->attr({xmlns =>
'http://www.service-now.com/'});

# get incident by sys_idmy@params=( SOAP::Data->name(categ
ory =>'Network'));

my%keyHash=%{$soap->call($method=>@params)->body->{'getRec
ordsResponse'}};

my$i=0;my$size=@{$keyHash{'getRecordsResult'}};for($i=0;$i
<$size;$i++) {my%record=%{$keyHash{'getRecordsResult'}}[$i]
;print"----- $i -----"
-----\n";foreach my$kk (keys%record) {print"$kk=$record
{$kk}\n";}}
```

deleteRecord

```
#!/usr/bin/perl -w

#use SOAP::Lite ( +trace => all, maptype => {} );use SOAP:
::Lite;

sub SOAP::Transport::HTTP::Client::get_basic_credentials{r
eturn'itil'=>'itil';}

my$soap= SOAP::Lite->proxy('http://localhost:8080/glide/in
cident.do?SOAP');

my$method= SOAP::Data->name('deleteRecord')->attr({xmlns =
>'http://www.service-now.com/'});
```

```
# delete incident by sys_idmy@params=( SOAP::Data->name(sys_id =>'46f67787a9fe198101e06dfcf3a78e99'));

my$result=$soap->call($method=>@params);

print_fault($result);
print_result($result);

sub print_result {my($result)=@_;

    if($result->body&&$result->body->{ 'deleteRecordResponse' })
    {my%keyHash=%{$result->body->{ 'deleteRecordResponse' }};foreach my$k(keys%keyHash){print"name=$k      value=$keyHash{$k}\n";}}}

sub print_fault {my($result)=@_;

    if($result->fault){print"faultcode=".$result->fault->{ 'faultcode' }."\n";print"faultstring=".$result->fault->{ 'faultstring' }."\n";print"detail=".$result->fault->{ 'detail' }."\n";}}
```

Python web services client examples

Examples demonstrating an integration with a Python web services client.

Requirements

The following examples require the installation of the following Python modules:

- fpconst <http://pypi.python.org/pypi/fpconst/0.7.2>
- PyXML <http://pyxml.sourceforge.net/topics/>
- SOAPpy <http://pywebsvcs.sourceforge.net/>

insert

This is an example of inserting an incident.

```
#!/usr/bin/python
```

```
from SOAPPy import SOAPPProxy
import sys

def createincident (params_dict ):

    # instance to send to
    instance = 'demo'

    # username/password
    username = 'itil'
    password = 'itil'

    # proxy - NOTE: ALWAYS use https://INSTANCE.servi
    ce-now.com, not https://www.service-now.com/INSTANCE for w
    eb services URL from now on!
    proxy = 'https://%s:%s@%s.service-now.com/inciden
    t.do?SOAP' % (username , password , instance )
    namespace = 'http://www.service-now.com/'
    server = SOAPPProxy (proxy , namespace )

    # uncomment these for LOTS of debugging output #s
    erver.config.dumpHeadersIn = 1 #server.config.dumpHeadersO
    ut = 1 #server.config.dumpSOAPOut = 1 #server.config.dumpS
    OAPIIn = 1

    response = server.insert (impact = int (params_d
    ict [ 'impact' ] ) , urgency = int (params_dict [ 'urgency'
    ' ] ) , priority = int (params_dict [ 'priority' ] ) , cat
    egory =params_dict [ 'category' ] , location =params_dict
    [ 'location' ] , caller_id =params_dict [ 'user' ] , assig
    nment_group =params_dict [ 'assignment_group' ] , assigned
    _to =params_dict [ 'assigned_to' ] , short_description =pa
    rams_dict [ 'short_description' ] , comments =params_dict
    [ 'comments' ] )

    return response

values = { 'impact': '1' , 'urgency': '1' , 'priority'
: '1' , 'category': 'High' , 'location': 'San Diego' ,
'user': 'fred.luddy@yourcompany.com' , 'assignment_group'
: 'Technical Support' , 'assigned_to': 'David Loo' , 'sh
ort_description': 'An incident created using python, SOAP
py, and web services.' , 'comments': 'This a test making
```

```
an incident with python.\n Isn \t life wonderful?' }\n\nnew_incident_sysid =createincident (values )\n\nprint "Returned sysid: "+ repr (new_incident_sysid )
```

getKeys

This is an example of executing getKeys on the demo instance using basic authentication.

```
#!/bin/env python\n\n# use the SOAPpy module from SOAPpy import SOAPPProxy\n\nusername , password , instance = 'admin' , 'admin' , 'demo'\nproxy , namespace = 'https://username:password@www.servicenow.com/'+instance+ '/incident.do?SOAP' , 'http://www.servicenow.com/'\n\nserver = SOAPPProxy (proxy ,namespace )\nresponse = server. getKeys (category = 'Network' )\n\nprint response. sys_id. split ( ',' )
```

getRecords

In this example, we get an incident, querying for category == "Network" (with basic authentication).

```
#!/bin/env python\n\n# use the SOAPpy module from SOAPpy import SOAPPProxy\n\nusername , password , instance = 'admin' , 'admin' , 'demo'\nproxy , namespace = 'https://username:password@www.servicenow.com/'+instance+ '/incident.do?SOAP' , 'http://www.servicenow.com/'\n\nserver = SOAPPProxy (proxy ,namespace )\nresponse = server. getRecords (category = 'Network' )\n\nfor record in response:
```

```
for item in record:  
    print item
```

get

In this example, we get an incident record by `sys_id` (with basic authentication).

```
#!/bin/env python  
  
# use the SOAPpy module from SOAPpy import SOAPProxy  
  
username , password , instance = 'admin' , 'admin' , 'demo'  
proxy , namespace = 'https://username:password@www.service-now.com/'+instance+ '/incident.do?SOAP' , 'http://www.service-now.com/'  
  
server = SOAPProxy (proxy ,namespace )  
response = server.get (sys_id = '9c573169c61122870019329fff72400' )  
  
for each in response:  
    print each
```

Advanced

This is an example of advanced Python script that reads a log file for a keyword invalid spi and creates an ECC Queue record where the payload is set to an alert of XML format.

```
#!/bin/env python  
  
# kevin.pickard@service-now.com  
08.07.03           initial creation  
  
from SOAPpy import SOAPProxy  
from xml.dom.minidom import getDOMImplementation  
import sys , os , socket , pickle , re  
  
# instance to send to  
instance = 'demo'  
  
# username/pass  
username = 'admin'
```

20

```
password = 'admin'

# log file to watch
syslogfile = '/var/log/cisco.log.ksp'

# state file
statefile = '/tmp/syslog_ecc.state-test'

# ECC queue values
soapagent = 'SOAPpy'
ecctopic = 'PIX Error: '
eccname = 'Invalid SPI: '
eccsource = 'Syslog'

# regex string to match
matchstring = 'invalid spi'

try:
    state = open (statefile , 'r' )
    lastbyte = pickle. load (state )
    state. close ( ) except:
    lastbyte = 0

#print 'DEBUG: lastbyte = '+str(lastbyte)

try:
    log = open (syslogfile , 'ro' ) except:
    errortopic = 'Script Error'
    errorname = 'Unable to open log file '+syslogfile+
    '.'

    errorpayload = 'This message was generated due to
an error condition encountered in a script. The name of t
he script is '+ os. path. basename ( sys. argv [ 0 ] )+
    ' on server '+ socket. gethostname ( )+ '.'

    proxy  = 'https://'+username+ ':'+'password+ '@'+in
stance+ '.service-now.com/ecc_queue.do?SOAP'
    namespace  = 'http://www.service-now.com/'
    server  = SOAPPProxy (proxy , namespace )
    server. config. dumpSOAPOut = 1
    server. config. dumpSOAPIn = 1
    response  = server. insert (agent =soapagent , top
ic =errortopic , name =errorname , source = sys. argv [ 0
] , payload =errorpayload )
```

```
        sys. exit ( 1 )

if lastbyte  != 0:
    try:
        log. seek (lastbyte ) except IOError:
            pass

loglines =log. readlines ( )

lastbyte =log. tell ( )

log. close ( )

state = open (statefile , 'w' ) pickle. dump (lastbyte , s
tate )
state. close ( )

# regex out the line
matchedlines = [ ] for line  in loglines:
    if re. search (matchstring , line ) != None:
        matchedlines. append (line )

#print 'DEBUG: len->loglines = '+str(len(loglines)) #prin
t 'DEBUG: lastbyte = '+str(lastbyte) #print 'DEBUG: matche
dlines = '+str(matchedlines)

if len (matchedlines ) == 0:
    sys. exit ( 0 )

proxy  = 'https://'+username+ ':'+password+'@'+instance+
'.service-now.com/ecc_queue.do?SOAP'
namespace  = 'http://www.service-now.com/'

server  = SOAProxy (proxy , namespace ) #server.config.du
mpSOAPOut = 1 #server.config.dumpSOAPIn = 1

entrystosend = { } for line  in matchedlines:
    device =line. split ( ) [ 3 ]
    sourceip =line. split ( ) [- 1 ]
    entrystosend [sourceip ] = [device , line ]

for key ,value  in entrystosend. iteritems ( ):
    #impl=getDOMImplementation() #newdoc = impl.creat
```

```
eDocument(None, "log_line", None) #top_element = newdoc.documentElement  
#text = newdoc.createTextNode(value[1]) #top_element.appendChild(text)  
  
response = server.insert(agent=soapagent, top  
ic=ecctopic+value[0], name=eccname+key, source=ecc  
source, payload=value[1])
```

Web services C Sharp .NET end to end tutorial

Examples demonstrating how to use .NET to consume a ServiceNow web service.

This tutorial will show you how to configure ServiceNow correctly to receive a web service request from your .NET client, as well as how to consume our web services using C# .NET.

- [Configure C sharp with .NET](#)

Configure web services within to receive web service requests from a .NET client..

- [Call a web service in visual studio .NET](#)

Call a web service using Visual Studio 2008.

- [C Sharp integration results](#)

If you have followed the tutorial correctly, you should receive the result whether you used a Service Reference or a Web Reference.

- [Troubleshoot a null response in a C Sharp integration](#)

Receiving a null response from ServiceNow's web service.

Configure web services within to receive web service requests from a .NET client..

1. To configure web services within ServiceNow, access the **System Properties > Web Services** module.

This module displays the system properties that are specific to web services within your instance. For security reasons, you will want to make sure that you require basic authorization for incoming SOAP

requests. This ensures that only authenticated users will be able to make any web services calls, whether it be via web service import sets or inserting/deleting/querying via direct web services.

Web Services

Please edit your changes and press Save

Customization Properties for Web Services

Require basic authorization for incoming RSS requests

Yes | No

Require basic authorization for incoming SOAP requests

Yes | No

2. This next step is very important if you are using .NET as a client to connect to ServiceNow. You must set the elementFormDefault property to false.

This property defines how the WSDLs are qualified. Of course, if you do not consume our WSDL and just create the XML manually, then this property is irrelevant.

This property sets the elementFormDefault attribute of the embedded XML schema to the value of unqualified, if set to true. This attribute indicates whether or not locally declared elements must be qualified by the target namespace in an instance document. If the value of this attribute is 'unqualified', then locally declared elements should not be qualified by the target namespace. If the value of this attribute is 'qualified', then locally declared elements must be qualified by the target namespace. For compatibility with Clients generated from WSDL (.NET Web Reference, Axis2 stub, webMethods, etc.), set this value to false. This value defaults to true.

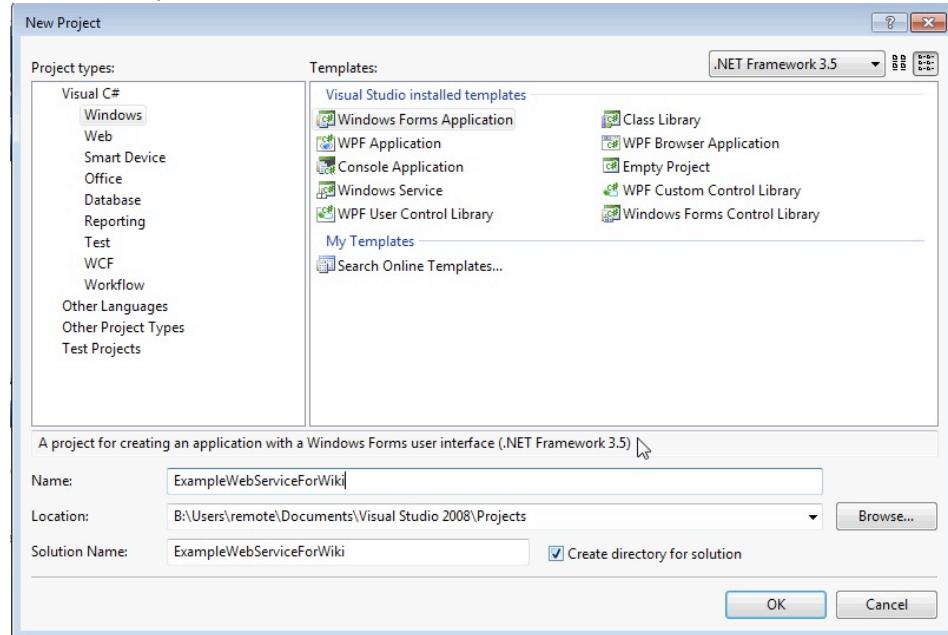
For further documentation, follow this URL http://wiki.service-now.com/index.php?title=Web_Services

Yes | No

Call a web service using Visual Studio 2008.

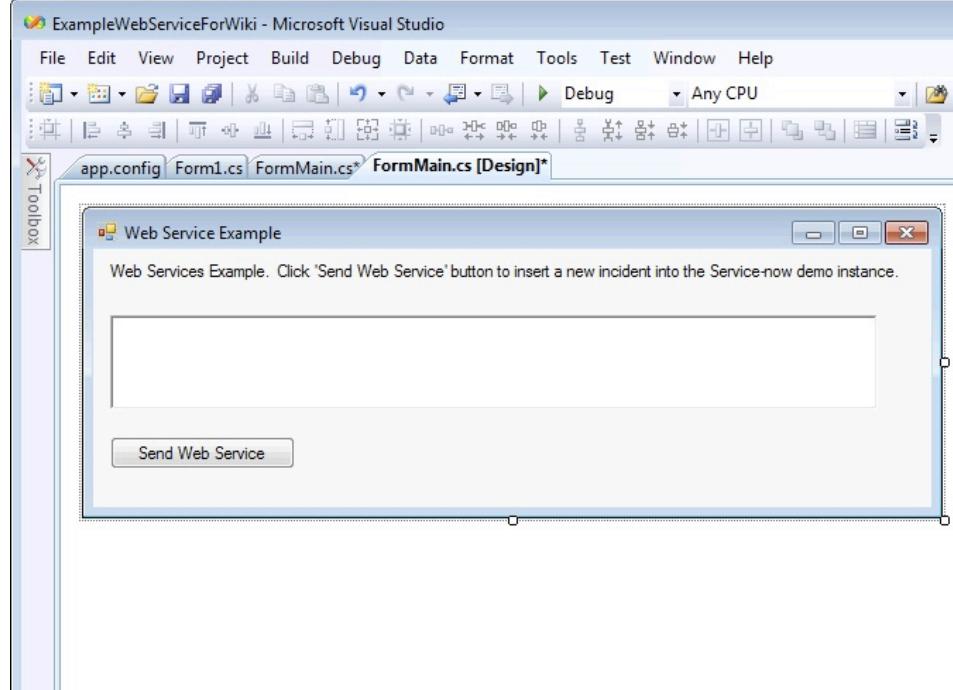
In this example, we will be using Visual Studio 2008. First, create a new Windows Form Application for this example.

Dot net project



On the resulting form, we created a richTextBox (which we named 'richTextBoxResult') and a button (named buttonResult).

Dot net example form



- Use a service reference in a C Sharp integration

Use a wizard to add a service reference for a C Sharp integration.

- Use a web reference in a C Sharp integration

Use a wizard to add a web reference for a C Sharp integration.

- C Sharp integration source code

After defining the source code, insert it.

Use a wizard to add a service reference for a C Sharp integration.

Go to the **Solutions Explorer** and select **Service References > Add Service Reference**. A wizard will appear asking for an address. Use: <https://<instance name>.service-now.com/incident.do?WSDL>. Accept the defaults for the rest of the wizard.

Open the app.config file and change the Security mode to "Transport" and the clientCredentialType and proxyCredentialType to "Basic"

Dot net app config

```
<security mode="Transport">
    <transport clientCredentialType="Basic" proxyCredentialType="Basic"
        realm="">^
        <extendedProtectionPolicy policyEnforcement="Never" />
    </transport>
    <message clientCredentialType="UserName" algorithmSuite="Default" />
</security>      [
```

Use a wizard to add a web reference for a C Sharp integration.

Go to the **Solutions Explorer** and select **Service References > Add Service Reference**. A wizard will appear. At the bottom of the form, there is an **Advanced** button. Click on it and click on the **Add Web Reference** button at the bottom of the new wizard page. This will start the **Web Reference** wizard. For the URL, use: `https://<instance name>.service-now.com/incident.do?WSDL` and name the web reference, 'WebReference1'. Accept the defaults for the rest of the wizard.

After defining the source code, insert it.

Now we are ready to insert the code. Double-click on the **Send Web Service** button on your form to open the backend code to the form that has been created. Here is the code to insert a record into the demo instance and to read the response.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ExampleWebServiceForWiki
{
    public partial class FormMain : Form
    {
        public FormMain()
        {
```

```
        InitializeComponent();
    }

    private void buttonSend_Click(object sender, EventArgs e)
    {
        /* SERVICE REFERENCE-SPECIFIC CODE
           ServiceReference1.ServiceNowSoapClient soapClient = new ServiceReference1.ServiceNowSoapClient();
           soapClient.ClientCredentials.UserName.UserName = "itil";
           soapClient.ClientCredentials.UserName.Password = "itil";

           ServiceReference1.insert insert = new ExampleWebServiceForWiki.ServiceReference1.insert();
           ServiceReference1.insertResponse response = new ExampleWebServiceForWiki.ServiceReference1.insertResponse();
           // END OF SERVICE REFERENCE CODE */

        // WEB REFERENCE-SPECIFIC CODE
        WebReference1.ServiceNow_incident soapClient = new ExampleWebServiceForWiki.WebReference1.ServiceNow_incident();
        System.Net.ICredentials cred = new System.Net.NetworkCredential("itil", "itil");
        soapClient.Credentials = cred;

        WebReference1.insert insert = new WebReference1.insert();
        WebReference1.insertResponse response = new WebReference1.insertResponse();
        // END OF WEB REFERENCE CODE

        insert.category = "Category";
        insert.comments = "Comments";
        insert.short_description = "My short description";

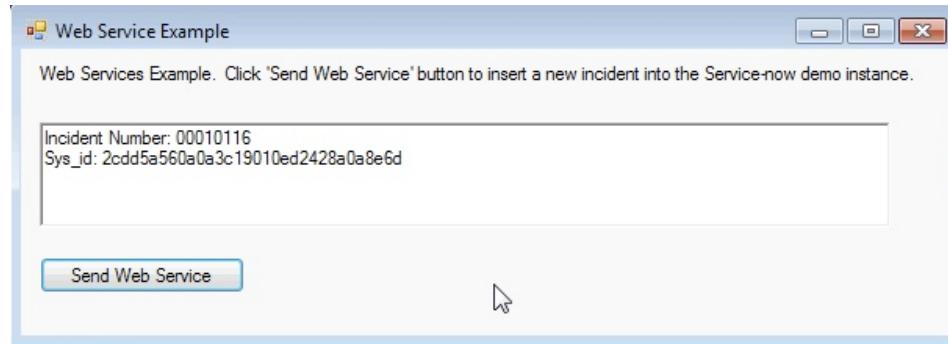
        try
        {
            response = soapClient.insert(insert);
            this.richTextBoxResult.Text = "Incident Nu
```

```
        mber: " + response.number + "\n";
                this.richTextBoxResult.Text += "Sys_id: "
+ response.sys_id;
            }
            catch (Exception error)
{
            this.richTextBoxResult.Text = error.Message;
}
}

}
```

If you have followed the tutorial correctly, you should receive the result whether you used a Service Reference or a Web Reference.

Dot net tutorial results



Receiving a null response from ServiceNow's web service.

If you are receiving a "null" response from your web service in your client code, then you may have missed the step in this tutorial for setting the elementFormDefault setting to "False".

Remember to recompile your code against the WSDL after you have changed this setting and saved it.

Retrieve a large number of records using SOAP

By default, a single SOAP request can retrieve a maximum of 250 records.

SOAP relies on Extensible Markup Language (XML) as its message format, and usually relies on other Application Layer protocols (most notably Remote Procedure Call (RPC) and HTTP) for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built.

Because of the verbose XML format, SOAP can be considerably slower than other transport methods. Therefore, sending a large amount of data via SOAP is inefficient and is discouraged. Because of this, ServiceNow has imposed a hard-limit of 250 records that can be retrieved at any time in a single query. You may find that this limit poses some technological challenges for your integration design.

- [SOAP strategies](#)

Retrieve the information that you need and make your integration more efficient.

Retrieve the information that you need and make your integration more efficient.

- [Use filters to limit the number of results](#)

One way to make your web service calls fit within the 250 record limit is to think about the design of your integrating application.

- [Use a local data store to pull data from](#)

If a large amount of data needs to be queried often, and the data does not need to be real-time, perform a sync of the ServiceNow table that you're interested in with your integrating application's data store.

- [Use Java/C#/PHP code to fetch the XML data using basic authentication](#)

If a local data store is not an option, another way to get the data is to call the CSV/XML processor directly and then parse the results.

One way to make your web service calls fit within the 250 record limit is to think about the design of your integrating application.

For example, let's assume that we are making an incident form in C# to show a user the incidents that are assigned to him.

Problematic query approach

The C# application makes a soap call to retrieve all of the incidents within ServiceNow. The application would then store the results locally in memory. When the user decides to view the incidents that are assigned to him, the application loops the internal array and displays the incidents that are assigned to the user.

A better query approach

The C# application makes a soap call to retrieve all of the incidents within ServiceNow that are assigned to the logged-in user. The results are stored locally in memory. When the user decides to view the incidents that are assigned to him, the application shows all the results to the user.

A performance-optimized query approach

The C# application makes no SOAP call initially. When a logged-in user decides to view the incidents that are assigned to him, the application presents him with the choice of viewing active, closed, etc. It gives him the ability to filter the results that he wants to see before the SOAP call is even made. Then, the user is only presented with the results that he wished to view.

If a large amount of data needs to be queried often, and the data does not need to be real-time, perform a sync of the ServiceNow table that you're interested in with your integrating application's data store.

Data push

- Using a scheduled job, ServiceNow can generate a csv/xml from a report and have it emailed to a specific location. The receiver might have a trigger to take the email attachment, parse it, and populate an internal table from which the application can communicate when the data is needed.

- Using a schedule job, ServiceNow can generate a csv/xml from a report and FTP it to a public FTP/FTPS location. The integrating product would consume this csv file on a regular basis and populate an internal table from which the application can communicate when the data is needed.

Note: Currently, the platform does not provide a method for extracting very large amounts of data and sending the output to an FTP server. However, a customization to perform that function is described at [here](#). The customization was developed for use in specific ServiceNow instances, and is not supported by ServiceNow Customer support. The method is provided as-is and should be tested thoroughly before implementation. Post all questions and comments regarding this customization to our community [forum](#).

Data pull

Using a cron job, a machine internal to your network can make a wget call to pull csv/xml data from any table within ServiceNow. The integrating product would consume this csv/xml file on a regular basis and populate an internal table from which the application can communicate when the data is needed. Examples of the wget command that would be used:

- wget --user=itil --password=itil --no-check-certificate https://<instance name>.service-now.com/incident_list.do?CSV
- wget --user=itil --password=itil --no-check-certificate https://<instance name>.service-now.com/incident_list.do?XML

If a local data store is not an option, another way to get the data is to call the CSV/XML processor directly and then parse the results.

Use the resulting data in a similar manner as you would a direct SOAP call. An example of this in PHP:

```
<?php  
//This example is in PHP  
  
$user = "itil";  
$pass = "itil";  
$userPass = $user.':'.$pass;
```

```
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, 'https://<instance name>.service-now.com/incident_list.do?CSV');
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC)
;
curl_setopt($ch, CURLOPT_USERPWD, $userPass);

$data = curl_exec($ch);
$info = curl_getinfo($ch);

if ($output === false) {
    $output = "No CURL data returned for $addr [". $info['http_code']. "]";
    if (curl_error($ch))
        $output .= "\n". curl_error($ch);
    print $output;
}
else{
    echo $data;
}
curl_close($ch);
?>
```

Outbound web services

You can use any of several outbound web services to integrate with ServiceNow applications, including REST and SOAP.

- [Outbound REST web service](#)

ServiceNow outbound REST functionality allows you to retrieve, create, update, or delete data on a web services server that supports the REST architecture.

- [Outbound SOAP web service](#)

The SOAP Message module can be used to develop, prototype, and save outbound SOAP messages that can be reused in business rules and scripts.

- [Outbound web services: Logging](#)

Log requests and responses for outbound web services such as REST and SOAP.

Outbound REST web service

ServiceNow outbound REST functionality allows you to retrieve, create, update, or delete data on a web services server that supports the REST architecture.

A REST message can be sent by a REST workflow activity or by using the RESTMessageV2 script API. You can run REST messages from a MID Server which allows the message to communicate with REST providers on an internal network.

ServiceNow REST functionality is flexible enough to accommodate many web service APIs. Be sure you are familiar with your web service and the parameters it accepts before attempting to define a REST message in ServiceNow.

- [REST message elements](#)

An outbound REST message is composed of several elements, such as the endpoint and HTTP methods.

- [Create a REST message](#)

You can send requests to a REST web service endpoint by creating a REST message record.

- [Outbound REST authentication](#)

Outbound REST messages support multiple types of authentication.

- [Variable substitution in outbound REST messages](#)

You can use variables when creating outbound REST messages and assign values to those variables when performing a request.

- [Scripting outbound REST](#)

You can send outbound REST requests from any place in the Now Platform where scripting is allowed.

REST message elements

An outbound REST message is composed of several elements, such as the endpoint and HTTP methods.

A REST message contains the following elements:

Elements

Element	Description
Endpoint	The endpoint is the URL of the data to be retrieved, updated, or deleted. Every REST message must specify an endpoint.
Headers	HTTP headers in REST messages contain information about the request, such as the desired response format. A REST message may specify any number of headers.
Authentication settings	Authentication settings include which type of authentication to use, such as basic auth or OAuth, as well as the credentials to use.
HTTP methods	<p>HTTP methods, such as GET, POST, or DELETE interact with the data at the endpoint.</p> <p>You can optionally override the parent REST message configuration in each HTTP method such as by specifying a different endpoint, authentication credentials, or headers.</p> <p>HTTP methods that send content, such as POST, include a message body detailing this content.</p>

Element	Description
	A REST message may specify multiple HTTP methods. When sending a REST message, such as through a workflow activity or script, you must specify which HTTP method to use.

Create a REST message

You can send requests to a REST web service endpoint by creating a REST message record.

Before you begin

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > REST Message**.
2. Click **New**.
3. Complete the following fields:

REST Message form

Field	Description
Name	Enter a descriptive name for this message.
Endpoint	Enter the endpoint that this REST message is sent to. The endpoint value may include variables using the format \${variable}.
Authentication type	Select the type of authentication to use, if any, and the profile record that contains the user credentials.

Field	Description
	<p>Outbound REST supports basic authentication and OAuth 2.0. Outbound REST supports mutual authentication with basic authentication only.</p> <p>Authentication configured here is inherited by the associated HTTP methods. You can configure authentication for each method which overrides any authentication setting at the message level.</p>
HTTP Headers	<p>Double-click a row in the HTTP Headers embedded list to define the header Name and Value.</p> <p>The web service provider determines which headers are supported or required.</p>

4. Click **Submit**.

After creating the REST message, a GET HTTP method is created automatically using the values from the REST message record.

What to do next

Create or edit HTTP methods and run a request.

- [Define a REST message HTTP method](#)

Define an HTTP method such as GET or POST to send a request to a web service provider.

- [Define a REST message HTTP header](#)

Define an HTTP header for a REST message or HTTP method to send that header with REST requests.

- [Sending outbound REST messages through a MID Server](#)

You can configure a REST message HTTP method to be sent through a MID Server.

- [Using special characters in URIs](#)

A REST function URI or function variable may use special characters, such as pipe (|) characters.

Define an HTTP method such as GET or POST to send a request to a web service provider.

Before you begin

Role required: web_service_admin

About this task

When you create a REST message record, several default HTTP methods are automatically created using settings inherited from the REST message record, such as the **Endpoint**. Subsequent changes to the REST message record are not applied to the HTTP methods automatically. You can create additional HTTP methods or modify the default HTTP methods to implement new behavior.

Procedure

1. Navigate to **All > System Web Services > Outbound > REST Message**.
2. Select a REST message you want to define an HTTP method for.
3. In the **HTTP Methods** related list, click **New**.
4. Select the **HTTP method** you want to use, such as GET or POST.
5. Enter the **Endpoint** this HTTP method should access.
The endpoint value may include variables using the format \$ {variable}.
6. Right-click the form header and select **Save**.

What to do next

After creating the HTTP method, you can override the security settings from the parent REST message, configure HTTP headers, add variables, or test the method. For PUT, POST, and PATCH methods you can define a message body.

- [Testing REST message HTTP methods](#)

After configuring an HTTP method for an outbound REST message, you can test it to ensure that the request is valid and the response returns as expected.

After configuring an HTTP method for an outbound REST message, you can test it to ensure that the request is valid and the response returns as expected.

To test an HTTP method, click the **Test** related link on the HTTP Method form.

Each test run displays the response status, such as 200 for a successful GET request, the full endpoint URL, any parameters passed in the request, and the response body.

Note: Fields on the Test Runs form are for information only; changes to these fields do not apply to the REST message or HTTP method. Do not modify these values when testing different REST message configurations. Instead, update the REST message or HTTP method, then run a new test.

If the HTTP method includes variables, the **Test value** for each variable in the **Variable Substitutions** related list is used when testing the method.

Completed test runs for an HTTP method appear in the **Test Runs** related list. If there was an error during the request, the **Error Code** and **Error Message** fields appear.

Define an HTTP header for a REST message or HTTP method to send that header with REST requests.

Before you begin

Role required: web_service_admin

About this task

You can specify an HTTP header for a REST message, or for an HTTP method. Headers defined for a REST message apply to all HTTP methods for that REST message. If you specify the same header for both a REST message and a child HTTP method, the value defined for the HTTP method overrides the value from the parent REST message.

Procedure

1. Navigate to **All > System Web Services > REST Message**.
2. Select a REST message.
3. (Optional) To specify a header for an HTTP method instead of the REST message, in the **HTTP Method** related list, select an HTTP method.
4. Select the **HTTP Request** tab.
5. In the **HTTP Headers** embedded list, click **Insert a new row**.
6. Enter the name of the header, such as Content-Type or Accept. Supported headers depend on the REST web service provider you want to connect to. Refer to the documentation for your web service provider to identify which headers are valid or required.
7. Click on the **Value** field for the new row and enter the value you want to assign this header. You can use a variable in the format \${variable} instead of a static value. You can assign a value to the variable when sending a REST request.
8. Click **Update**.

You can configure a REST message HTTP method to be sent through a MID Server.

By using a MID Server, the request can reach an endpoint that is behind a firewall or within a private network.

To configure an HTTP method to use a MID Server, select a MID Server in the **Use MID Server** field on the HTTP Method form. The instance must have an active MID Server to use this functionality.

A REST function URI or function variable may use special characters, such as pipe (|) characters.

When using these characters in a REST message, use URL encoding to escape these characters. For example, to use a parameter value of user | title, enter user%7Ctitle. Entering special characters directly may cause the REST message to fail and display the response Invalid uri <URL>: Invalid query.

Outbound REST authentication

Outbound REST messages support multiple types of authentication.

Different web service providers may require a specific type of authentication. Outbound REST supports the following authentication formats.

- Basic authentication using a username and password
- OAuth 2.0 using an OAuth provider and profile
- Mutual authentication using protocol profiles

Overriding REST authentication

You can define authentication for a REST message, or individually for each HTTP method. HTTP methods inherit authentication from their parent REST message record when the HTTP method **Authentication type** is **Inherit from parent**, which is the default value.

You can disable authentication for a specific HTTP method by setting the **Authentication type** field to **No authentication**, or specify authentication that is different from the parent REST message by selecting basic auth or OAuth.

Authentication requirements

Outbound REST supports mutual authentication only when using basic authentication. Mutual authentication is not available with OAuth 2.0.

OAuth 2.0 can be used only with messages that are not configured to use a MID Server. You cannot send OAuth 2.0 authenticated messages through a MID Server.

When scripting new REST messages configured with authentication you must use the RESTMessageV2 API. The legacy RESTMessage APIs do not support current authentication formats.

- [Configure a REST message with basic auth](#)

You can configure an outbound REST message to provide basic authentication credentials with each request.

- [Configure a REST message with OAuth](#)

You can configure an outbound REST message to send OAuth credentials with the request.

- [Outbound REST mutual authentication](#)

Mutual authentication causes the web service provider and consumer to authenticate with each other before communicating.

You can configure an outbound REST message to provide basic authentication credentials with each request.

Before you begin

Role required: `web_service_admin`

Before starting this procedure, ensure there is a REST Message record that you want to configure to use basic auth.

Note: Ensure any scripts that call this REST message use the RESTMessageV2 API. The RESTMessageV2 API is required to send authenticated REST messages via scripts.

Procedure

1. Navigate to **All > System Web Services > Outbound > REST Message**.
2. Select a REST message record.
3. In the **Authentication type** field, select **Basic**.

Note: The **Basic (Simple)** choice appears on REST message records configured to use basic authentication prior to the Geneva release. This choice is intended for compatibility with older REST messages and should not be used for REST messages created in the Geneva release or later.

4. In the **Basic auth profile** field, select the basic authentication profile that contains the credentials you want to send.
5. Click **Submit**.

What to do next

Test the REST message to ensure you receive the expected response. You can optionally specify different authentication settings for each HTTP method related to this REST message, overriding the parent REST message settings.

- [Create a basic auth profile](#)

Create a basic auth profile to specify basic authentication credentials for one or more REST messages.

Create a basic auth profile to specify basic authentication credentials for one or more REST messages.

Before you begin

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > REST Message**.
2. Select a REST message record.
3. In the **Authentication type** field, select **Basic**.
4. In the **Basic auth profile** field, click the reference lookup icon.
5. Click **New**.
6. Enter a descriptive **Name** for the profile.

7. Enter the **Username** and **Password** you want to send as basic authentication credentials.
8. Click **Submit**.

What to do next

Configure a REST message to use this basic auth profile.

You can configure an outbound REST message to send OAuth credentials with the request.

Before you begin

Role required: web_service_admin and oauth_admin

Before starting this procedure, ensure:

- There is a REST Message record that you want to configure to use OAuth.
- There is an OAuth provider set up in the OAuth application registry with the OAuth client information to use.
- The OAuth provider has an associated OAuth 2.0 profile.
- The REST message HTTP Methods are not configured to use a MID Server.

Note: Ensure any scripts that call this REST message use the RESTMessageV2 API. The RESTMessageV2 API is required to send authenticated REST messages via scripts.

Procedure

1. Navigate to **All > System Web Services > Outbound > REST Message**.
2. Select a REST message record.
3. In the **Authentication type** field, select **OAuth 2.0**.
4. In the **OAuth profile** field, select the OAuth 2.0 profile that specifies the credentials you want to send.

5. Right-click the form header and select **Save**.
An info message appears at the top of the form indicating that you must request a new OAuth token.
6. Click the **Get OAuth Token** related link.
Depending on your OAuth provider, a separate window from your provider may appear asking for confirmation before providing a token. Complete any steps required by the provider to obtain the token.

What to do next

Test the REST message to ensure you receive the expected response. You can optionally specify different authentication settings for each HTTP method related to this REST message, overriding the parent REST message settings.

- [Use a third-party OAuth provider](#)

Each client application must register with the instance to participate in OAuth authorization.

- [Outbound REST with OAuth 2.0 profile tutorial - integrating with Google Contacts API](#)

This tutorial explains how to use an OAuth 2.0 profile to authenticate an outbound REST message with Google to retrieve contact information.

Each client application must register with the instance to participate in OAuth authorization.

Before you begin

Role required: admin

About this task

Note: ServiceNow only supports third-party OAuth providers to authorize requests from ServiceNow to third-party APIs.

For example, you might want to integrate with a third-party Calendar service which requires OAuth 2.0 access tokens to read a user's scheduled events and create events. Configure the Calendar service

as a third-party OAuth provider. This configuration allows you to get an access token from the Calendar service and then use the token to make requests against the service from ServiceNow.

Procedure

1. Navigate to **All > System OAuth > Application Registry** and then click **New**.
2. On the interceptor page, click **Connect to a third party OAuth Provider** and then fill in the form.

Field	Description
Name	A unique name that identifies the application to access.
Client ID	The unique ID of the application. The instance uses the client ID when requesting an access token. You must get the client ID from the authorization server.
Client Secret	[Required] The shared secret string that the instance and the application use to authorize communications with one another. If you do not enter the correct client secret, tokens are not issued.
OAuth API Script	This option enables you to reference an amended OAuthUtil script include. Copy and rename the default OAuthUtil script include file, and then amend this version for specific requests and responses to match your 3rd party OAuth provider. The amended script name must have the prefix OAuth . See OAuthUtil to add the

Field	Description
	required body parameter in the proper method.
Logo URL	The URL that contains an image to use as the application logo.
Default Grant Type	<p>The type of grant:</p> <ul style="list-style-type: none">• Authorization code: The code that is granted to the client to obtain an access token, which is then used to obtain access to the resource. If you select this option, then you need an authorization URL (the URL of the authorization server).• Resource owner password credentials: The user name and password of the user that is trying to obtain access to the resource.• Client Credentials: The client ID and client secret, which are both used to get the access token. This method does not provide refresh tokens.• JWT Bearer: An authorization server validates a JWT token which enables identity and security information to be shared across security domains.• SAML2 Bearer: Generates the SAML2 assertion and then exchanges the

Field	Description
	<p>assertion for the access tokens with the provider.</p> <p>Note: For outbound request to SuccessFactors use the SAML2 Bearer as the Default Grant Type.</p>
Refresh Token Lifespan	The refresh token lifespan in seconds.
Accessible from	The application scope that this registry is accessible from.
Active	A check box that indicates that the application registry is active.
Authorization URL	If you are using the authorization code grant type, the URL of the endpoint to authorize the user. If you are accessing another ServiceNow instance, append /oauth_auth.do to the URL.
Token URL	The location of the token endpoint that the instance uses to retrieve and refresh tokens. If you are accessing another ServiceNow instance, append /oauth_token.do to the URL.
Redirect URL	The application endpoint that receives the authorization code. Leave the field empty to have the instance auto-generate the URL. If you are accessing another ServiceNow instance, append /oauth_redirect.do to the URL.

Field	Description
Token Revocation URL	The location of the endpoint that the instance uses to revoke the token. If you are accessing another instance, append /oauth_revoke.do to the URL.
Comments	Additional information to associate with the application.
Embedded lists	
OAuth Entity Profiles	The profiles that are associated with the OAuth provider. The profile includes the grant type. Click the profile name to go to the OAuth Entity Profile form .
OAuth Entity Scopes	The entity scopes associated with the OAuth provider. The scope identifies the services the application has access to. Click the scope name to go to the OAuth Entity Scope form .

3. Click **Submit**. The record is saved in the Application Registries [oauth_entity] table.
The system creates a record in the Application Registries [oauth_entity] table of type OAuth Provider. The instance also auto-generates a default profile using the specified grant type, but without any scopes.

What to do next

You can [create additional profiles](#), each with scopes.

- [OAuth profiles and scopes](#)

In the OAuth provider scenario, profiles and scopes specify the grant type, authorization type, and level of access.

In the OAuth provider scenario, profiles and scopes specify the grant type, authorization type, and level of access.

In the OAuth provider scenario, the OAuth profile refers to a combination of a grant type and at least one scope. The scope specifies the access that the user has to the protected resource, such as **read** or **write**. You can create a profile for each third-party provider and obtain the specific set of scopes from the provider. See [Specify an OAuth profile](#) and [Specify an OAuth scope](#) for more information. The instance also uses OAuth profiles when a REST call specifies OAuth 2.0 authentication. The instance auto-creates a default profile for each third-party provider record that you create. There can be only one default profile.

Specify the following parameters, which are saved in the OAuth Requestor Profile [OAuth_requestor_profile] table:

OAuth parameters for default profile support

Parameter	Description
oauth_requestor	The sys_id of the object, which can be a user record or an email account.
oauth_requestor_context	Descriptor that provides context for the OAuth requestor. As a good practice, use the name of the table where the oauth_requestor object is saved.
oauth_provider_profile	The sys_id of the OAuth profile record that is the default (see Specify an OAuth profile).

When the user attempts to authenticate, the provider accesses the OAuth Requestor Profile table to look for the user. If the user is found, the authentication is successful. If not, the provider accesses the default profile to determine the grant type and how to proceed with the authentication.

- [Specify an OAuth profile](#)

An OAuth profile includes the grant type that the third-party OAuth provider needs to obtain access to the restricted resource.

- **Specify an OAuth scope**

Specify the OAuth scopes that you get from the provider. Scopes can be any level of access specified by the provider, such as read, write, or any string, including a URL.

An OAuth profile includes the grant type that the third-party OAuth provider needs to obtain access to the restricted resource.

Before you begin

Role required: admin

About this task

Procedure

1. Open a third-party OAuth provider record.
2. In the OAuth Entity Profiles embedded list, click **Insert a new row** and then enter a name for the profile.
3. Right-click the Application Registry form header and select **Save**. The system creates the profile record.
4. Click the name of the profile you created and then fill in the form fields.

OAuth Entity Profile Scopes	
<input checked="" type="radio"/> OAuth Entity Scope	User Info
<input type="radio"/> User Info	
Insert a new row...	

Field	Description
Name	Enter a descriptive name.
OAuth provider	Verify the provider that is associated with the profile.
Grant type	Select the grant type: <ul style="list-style-type: none">• Authorization code: The code that is granted to the client to obtain an access token, which is then used to obtain access to the resource. If you select this option, then you need an authorization URL (the URL of the authorization server).• Resource owner password credentials: The user name and password of the user that is trying to obtain access to the resource.• Client Credentials: The client ID and client secret, which are both used to get the access token. This method does not provide refresh tokens.• JWT Bearer: An authorization server validates a JWT token which enables identity and security information to be shared across security domains.• SAML2 Bearer: Generates the SAML2 assertion and then exchanges the

Field	Description
	assertion for the access tokens with the provider. Note: For outbound request to SuccessFactors use the SAML2 Bearer as the Default Grant Type.
Is default	Select this option to make the profile the default option for the associated provider.
Embedded list	
OAuth Entity Profile Scopes	Specify the OAuth entity scope.

5. Click **Update**.

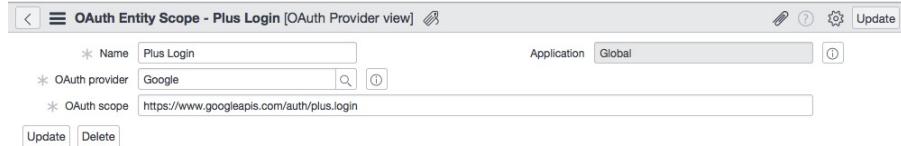
Specify the OAuth scopes that you get from the provider. Scopes can be any level of access specified by the provider, such as read, write, or any string, including a URL.

Before you begin

Role required: admin

Procedure

1. Open a third-party OAuth provider record.
2. Open a profile associated with the provider.
3. In the OAuth Entity Profile Scopes embedded list, click **Insert a new row** and then enter a **Name** for the profile.
4. Right-click the **OAuth Entity Profile** form header and select **Save**. The profile record is created.
5. Click the name of the scope that you created and then fill in the form fields.



The screenshot shows a ServiceNow form titled "OAuth Entity Scope - Plus Login [OAuth Provider view]". The form has three required fields: "Name" (Plus Login), "OAuth provider" (Google), and "OAuth scope" (https://www.googleapis.com/auth/plus.login). There are also buttons for "Update" and "Delete".

OAuth Entity Scope form fields

Field	Description
Name	Enter a descriptive name.
OAuth provider	Verify the provider associated with this scope.
OAuth scope	The scope that you are granted by the provider. Typical scopes are read and write. Scopes can be any string that the provider specifies.

6. Click **Update**.

This tutorial explains how to use an OAuth 2.0 profile to authenticate an outbound REST message with Google to retrieve contact information.

The procedures detailed in this tutorial require the oauth_admin and web_service_admin roles. Ensure you have both of these roles before starting this tutorial.

1. OAuth 2.0 tutorial - configure the Google service as an OAuth provider

Use the Google Developer Console to set up an OAuth 2.0 provider.

2. OAuth 2.0 tutorial - create an OAuth provider and profile

Set up the Google service as an OAuth provider in ServiceNow by entering your client information, Google API URLs, and configuring the OAuth profile.

3. OAuth 2.0 tutorial - create a REST message

Create a REST message and associated HTTP method to contact the Google service using the OAuth 2.0 profile.

Use the Google Developer Console to set up an OAuth 2.0 provider.

Before you begin

Role required: None

This procedure is performed within the Google Developer Console. You must have a Google account to access this console.

About this task

Configure the Google service in order to obtain a client ID and client secret, and specify your ServiceNow instance URL as the OAuth redirect URL.

Note: This information describes the state of the Google Developer Console and Contacts API as of July 22, 2015. Changes made after that date may not be included in this document.

Procedure

1. Navigate to the Google Developer Console (<https://console.developers.google.com>).
2. Log in using your Google credentials.
3. Click **Select a project**.
4. Click **Create a project**.
5. Enter a **Project name**.
6. Click **Create**.
After Google creates the project, the project dashboard appears.
7. Navigate to **APIs & auth > APIs**.
8. Select the Contacts API.
9. Click **Enable API**.

10. Navigate to **APIs & auth > Credentials**.
11. Click **Create new Client ID**.
12. Ensure the **Web application** radio button is selected and click **Configure consent screen**.
13. Enter a descriptive **Product name**.
This name appears when you authorize the OAuth token in your instance.
14. Click **Save**.
15. In the **Create Client ID** window, add the OAuth redirect URI for your instance to the **Authorized redirect URIs** field.
This URI follows the format `https://<instance>.service-now.com/oauth_redirect.do`
16. Click **Create Client ID**.
The client ID information appears.
17. Record the **Client ID** and **Client secret** values.
You will need these values to configure the Google service as an OAuth provider in your instance.

Set up the Google service as an OAuth provider in ServiceNow by entering your client information, Google API URLs, and configuring the OAuth profile.

Before you begin

Role required: oauth_admin

You must have configured the Google service as an OAuth provider and recorded your **Client ID** and **Client Secret** values.

Procedure

1. Navigate to **All > System OAuth > Application Registry**.
2. Click **New**.
3. Select **Connect to a third party OAuth Provider**.
4. Enter a **Name** for the OAuth provider. For this example, use **Google**.

5. Enter the **Client ID** and **Client Secret** that you obtained from Google.
6. Set the **Default Grant type** to **Authorization Code**.
7. In the **Authorization URL** field, enter `https://accounts.google.com/o/oauth2/auth`.
8. In the **Token URL** field, enter `https://www.googleapis.com/oauth2/v3/token`.
9. In the **Redirect URL** field, enter `https://<instance>.service-now.com/oauth_redirect.do`.
This URL must match the redirect URL provided to Google.
10. In the **Token Revocation URL** field, enter `https://accounts.google.com/o/oauth2/revoke`.
11. Right-click the form header and select **Save**.
A new OAuth Entity Profile record is created.
12. In the **OAuth Entity Scopes** embedded list, add a new row with the **Name** and **OAuth scope** values set to `https://www.googleapis.com/auth/contacts.readonly`.
13. Right-click the form header and select **Save**.
14. In the **OAuth Entity Profiles** embedded list, select the automatically-created profile.
15. In the **OAuth Entity Profile Scopes** embedded list, add a new row and select the Google contacts API read-only scope.
16. Click **Update**.

Create a REST message and associated HTTP method to contact the Google service using the OAuth 2.0 profile.

Before you begin

Role required: web_service_admin and oauth_admin

You must have configured an OAuth provider and profile using the Google API information and your OAuth credentials.

Procedure

1. Navigate to **All > System Web Services > REST Message**.
2. Click **New**.
3. Enter a descriptive **Name**.
4. In the **Endpoint** field, enter `https://www.google.com/m8/feeds/contacts/default/full`.
By using default instead of a specific username, the Google API uses the OAuth credentials to determine which account to get information from.
5. In the **Authentication** tab, set the **Authentication type** to **OAuth 2.0**.
6. In the **OAuth profile** field, select the Google contacts OAuth profile.
7. Right-click the form header and select **Save**.
8. Click the **Get OAuth Token** related link to request an authorization token from Google using the configured client ID and secret.
9. In the Request for Permission window that appears, click **Accept** to grant access to your Google contacts.
The token aquired is not directly accessible in your instance.
10. In the **HTTP Methods** related list, select the **GET** method.
11. Leave the HTTP method **Authentication type** as **-- None --** to use the OAuth profile from the parent REST message record.
12. On the **HTTP Request** tab, add a new row to the **HTTP Headers** related list with a **Name** of `GData - Version` and a **Value** of `3.0`.
13. Right-click the form header and select **Save**.
14. Click the **Test** related link.
The test result should display an **HTTP Status** of 200, and the result of the contacts API call.

Mutual authentication causes the web service provider and consumer to authenticate with each other before communicating.

Mutual authentication is not available for outbound web services that use a MID Server.

Note: For information about mutual authentication for inbound web services, see [Certificate-based authentication](#).

Variable substitution in outbound REST messages

You can use variables when creating outbound REST messages and assign values to those variables when performing a request.

Variables are allowed in the **Endpoint** URL, HTTP Header and HTTP Query Parameter **Value** fields, and the **Content** field for POST and PUT methods.

The syntax for variables is \${variable_name}. The REST message substitutes this variable with the parameter values provided when the method runs. For example, if the REST message **Endpoint** is http://myserver.mycompany.com/offices/\${id}, a parameter named id must exist and contain a value that can be used when the method runs.

You can assign a value to variables when running the request using the RESTMessageV2 API setStringParameter and setStringParameterNoEscape methods.

When testing an HTTP method that includes variables, the **Test value** for each variable in the **Variable Substitutions** related list is used.

- [Generate REST message variables](#)

You can automatically populate the list of variable substitutions, based on variables defined in several REST message HTTP method fields.

You can automatically populate the list of variable substitutions, based on variables defined in several REST message HTTP method fields.

Before you begin

Role required: web_service_admin

Before starting this procedure, create a REST Message record with at least one HTTP method that uses variables.

Procedure

1. Navigate to **All > System Web Services > REST Message**.

2. Select a REST Message record.
3. Select a method from the **HTTP Methods** related list.
4. Click the **Auto-generate variables** related link.

The **Variable Substitutions** related list is automatically populated for any variables defined in the HTTP Method **Endpoint** field and the **HTTP Headers** and **HTTP Query Parameters** embedded lists. For POST and PUT messages, variables defined in the **Content** field are also used.

What to do next

You can use the REST Message workflow activity to send the message, or click **Preview Script Usage** to get a sample script. The sample script includes a setStringParameter call for each defined variable substitution that allows you to assign a value to the variable in your script.

Scripting outbound REST

You can send outbound REST requests from any place in the Now Platform where scripting is allowed.

For example, you can return data from a REST endpoint using a business rule when an event is triggered. Create a script from scratch or let the REST message preview feature create the script based on content and parameters you provide in the method record.

For detailed API information about the server-side RESTMessageV2 and RESTResponseV2 APIs, see the API documentation on developer.servicenow.com. For additional tips and best practices, see the [Outbound REST Web Services RESTMessageV2 and SOAPMessageV2 execute\(\) vs executeAsync\(\) Best Practices \[KB0694711\]](#) article in the Now Support Knowledge Base.

- Generate a REST message script preview

You can generate an example script to send a REST message based on content and parameters you provide in the method record.

- Direct RESTMessageV2 example

You can send an outbound REST message directly to the endpoint.

- RESTMessageV2 MID server example

You can send an outbound REST message through a MID Server.

- [Recordless RESTMessageV2 example](#)

You can use the RESTMessageV2() constructor with no parameters to define a REST message entirely in the script.

You can generate an example script to send a REST message based on content and parameters you provide in the method record.

Before you begin

Role required: web_service_admin or admin

About this task

Generate an example script and use it as a starting point when scripting outbound REST messages.

Procedure

1. Navigate to **All > System Web Services > REST Message**.
2. Select a REST message record.
3. In the **HTTP Methods** related list, select an HTTP method record.
4. Ensure the HTTP method is configured as needed, including any variables.
5. Save the record.
6. In the Variable Substitutions related list, assign a value to each variable.
7. Under Related Links, click **Preview script usage**.
The instance displays the script that the REST message generated for this method.

Preview REST Message script usage 

```
try {
    var r = new sn_ws.RESTMessageV2('Yahoo Finance', 'get');

    //override authentication profile
    //authentication type ='basic'/ 'oauth2'
    //r.setAuthentication(authentication type, profile name);

    var response = r.execute();
    var responseBody = response.getBody();
    var httpStatus = response.getStatusCode();
}
catch(ex) {
    var message = ex.getMessage();
}
```

8. Copy this script and modify it as needed to use elsewhere in the instance.

What to do next

Refer to the RESTMessageV2 and RESTResponseV2 APIs for more information on available scripting methods. Outbound REST scripting examples are also available.

You can send an outbound REST message directly to the endpoint.

In this example, the script sends a REST message requesting a stock quote and waits for a response. If there is no response from the web service provider, or if the specified REST message record is unavailable, the script throws an error, handled in this example by the try-catch block.

```
var requestBody;
var responseBody;
var status;
var sm;
try{
    sm = new sn_ws.RESTMessageV2("Yahoo Finance", "get");
    // Might throw exception if message doesn't exist or
    // not visible due to scope.
    sm.setBasicAuth("admin","admin");
    sm.setStringParameter("symbol", "NOW");
```

```
        sm.setStringParameterNoEscape("xml_data","<data>te
st</data>");
        sm.setHttpTimeout(10000); //In milliseconds. Wait
at most 10 seconds for response from http request.

        response = sm.execute(); //Might throw exception if
http connection timed out or some issue with sending requ
est itself because of encryption/decryption of password.
        responseBody = response.haveError() ? response.get
ErrorMessage() : response.getBody();
        status = response.getStatusCode();
    } catch(ex) {
        responseBody = ex.getMessage();
        status = '500';
    } finally {
        requestBody = sm ? sm.getRequestBody():null;
    }
    gs.info("Request Body: " + requestBody);
    gs.info("Response: " + responseBody);
    gs.info("HTTP Status: " + status);
```

You can send an outbound REST message through a MID Server.

By sending the message through a MID Server, you can access endpoints that are behind a firewall or within a private network. All REST messages sent through a MID Server are asynchronous.

Example

```
var requestBody;
var responseBody;
var status;
var sm;
try{
    sm = new sn_ws.RESTMessageV2("Yahoo Finance", "get
"); // Might throw exception if message doesn't exist or
not visible due to scope.
    sm.setBasicAuth("admin","admin");
    sm.setStringParameter("symbol", "NOW");
    sm.setStringParameterNoEscape("xml_data","<data>te
st</data>");
    sm.setMIDServer('mid_server_name');
    response = sm.executeAsync(); // Might throw exce
ption if http connection timed out or some issue with sendi
```

```
ng request itself because of encryption/decryption of password.

    response.waitForResponse(60); // In seconds. Wait at most 60 seconds to get response from ECC Queue/Mid Server //Might throw exception timing out waiting for response in ECC queue.

    responseBody = response.haveError() ? response.getErrorMessage() : response.getBody();
    status = response.getStatusCode();
} catch(ex) {
    responseBody = ex.getMessage();
    status = '500';
} finally {
    requestBody = sm ? sm.getRequestBody():null;
}
gs.info("Request Body: " + requestBody);
gs.info("Response: " + responseBody);
gs.info("HTTP Status: " + status);
```

Note: This example uses `waitForResponse` to pause for a response, and then details how the response is handled. However, when using `executeAsync`, consider processing the response body in a separate business rule to take advantage of the asynchronous call rather than using `waitForResponse`.

You can use the `RESTMessageV2()` constructor with no parameters to define a REST message entirely in the script.

When using this constructor you must provide an endpoint and HTTP method. In this example, the script creates an empty REST message and sets the values needed to insert an incident record.

```
var restMessage = new sn_ws.RESTMessageV2();
restMessage.setBasicAuth("admin", "admin");
restMessage.setHttpMethod("post");
restMessage.setEndpoint("http://<instance>.service-now.com/api/now/table/incident");
restMessage.setRequestBody("{\"short_description\" : \"Test incident\"}");
var response = restMessage.execute();
```

Outbound SOAP web service

The SOAP Message module can be used to develop, prototype, and save outbound SOAP messages that can be reused in business rules and scripts.

You can use outbound SOAP messages in scripts using the SOAPMessageV2 API and the SOAPResponseV2 API. Examples detailing how to script outbound SOAP are available.

- [Outbound SOAP video tutorial](#)

The following video tutorial demonstrates how to configure outbound SOAP web service messages to consume third-party web services from an instance.

- [SOAP message](#)

Information needed to send SOAP requests is stored in SOAP message records.

- [Connectivity details](#)

For the ServiceNow-initiated SOAP requests to successfully communicate with the web service provider inside a remote network, the ServiceNow instance must have HTTP or HTTPS access to the SOAP endpoint at the provider.

- [Outbound SOAP security](#)

You can authenticate outbound SOAP messages using several different security protocols.

- [Configure SOAP with a proxy](#)

Certain properties provide support for SOAP requests to use a web proxy server.

- [Scripting outbound SOAP](#)

You can send outbound SOAP requests from any place in the Now Platform where scripting is allowed.

Outbound SOAP video tutorial

The following video tutorial demonstrates how to configure outbound SOAP web service messages to consume third-party web services from an instance.

SOAP message

Information needed to send SOAP requests is stored in SOAP message records.

Each record specifies an endpoint for the request, the required format of the request as a web services description language (WSDL) file, authentication information, and a list of functions that can run against the endpoint.

- [Create a SOAP message](#)

Create a SOAP message to define the remote endpoint, WSDL, and authentication settings.

- [SOAP message functions](#)

After you create a SOAP message record, you can click **Generate sample SOAP messages** to populate the SOAP Message Functions related list.

- [Variable substitution in outbound SOAP](#)

To use variable substitution, use the format \${<variable_name>} instead of defining a specific value.

- [Test the SOAP message](#)

Test a SOAP message to validate the configuration before using the message in an integration.

- [Send a SOAP message through a MID server](#)

When creating SOAP message functions, you can configure the function to be sent through a MID Server.

- Create a SOAP message from a WSDL that references an external XSD file

Create a SOAP message from a WSDL and external XSD file.

Create a SOAP message to define the remote endpoint, WSDL, and authentication settings.

Before you begin

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > SOAP Message**.
2. Click **New**.
3. Enter a Name to identify the SOAP message.
4. Specify a WSDL using one of these options:
 - To download and use an online WSDL source, select the **Download WSDL** check box and enter the URL for the WSDL in the WSDL field.
 - To enter the WSDL directly, clear the **Download WSDL** check box, and then copy and paste the WSDL XML into the WSDL XML field.
5. If the endpoint is protected by basic authentication, select the **Use basic auth** check box and enter the credentials.
6. If the endpoint requires mutual authentication, select the **Enable mutual authentication** check box and select a Protocol profile to use for mutual authentication.
7. Click **Submit**.

This image shows an example of a SOAP message that connects to a demo instance of ServiceNow.

SOAP Message Required field

Name:	demo1 incident	Created:	2010-03-31 17:30:58	Edit
WSDL:	https://demo1.service-now.com/incident.do?WSDL	Created by:	dloo	Edit
Download WSDL:	<input checked="" type="checkbox"/>	Use basic auth: <input type="checkbox"/>		
Description:	Sample Service-now direct web service			
WSDL XML	<pre><?xml version="1.0" encoding="UTF-8"?> <wsdl:definitions targetNamespace="http://www.service-now.com" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:tns="http://www.service-now.com/incident" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:nsns="http://www.service-now.com" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"></pre>			
Update	Delete			

Related Links

[Generate sample SOAP messages](#)

SOAP Message Functions [New](#) [SOAP message = demo1 incident](#)

		SOAP action		Envelope		Lock		Updated
<input type="checkbox"/>		insert	http://www.service-now.com/incident/insert	<SOAP-ENV:Envelope xmlns:xsd="http://www...	true	2010-04-05 10:23:49		
<input type="checkbox"/>		update	http://www.service-now.com/incident/update	<SOAP-ENV:Envelope xmlns:xsd="http://www...	false	2010-06-02 15:57:57		
<input type="checkbox"/>		deleteMultiple	http://www.service-now.com/incident/dele...	<SOAP-ENV:Envelope xmlns:xsd="http://www...	false	2010-06-02 15:57:58		
<input type="checkbox"/>		getKeys	http://www.service-now.com/incident/getK...	<SOAP-ENV:Envelope xmlns:xsd="http://www...	false	2010-06-02 15:57:59		
<input type="checkbox"/>		deleteRecord	http://www.service-now.com/incident/dele...	<SOAP-ENV:Envelope xmlns:xsd="http://www...	false	2010-04-05 10:22:50		
<input type="checkbox"/>		get	http://www.service-now.com/incident/get	<SOAP-ENV:Envelope xmlns:xsd="http://www...	false	2010-04-05 10:22:50		
<input type="checkbox"/>		getRecords	http://www.service-now.com/incident/getR...	<SOAP-ENV:Envelope xmlns:xsd="http://www...	false	2010-06-02 15:58:00		

[Actions on selected rows...](#)

After you create a SOAP message record, you can click **Generate sample SOAP messages** to populate the SOAP Message Functions related list.

The instance creates these functions by reading the supplied WSDL definition.

Soap message function

SOAP Message Function

Function:		insert	SOAP message:	demo1 incident	Update	Delete		
Basic auth user ID:	til		Lock:	<input checked="" type="checkbox"/>				
Basic auth user password:	*****		Use basic auth:	<input checked="" type="checkbox"/>				
SOAP action:	http://www.service-now.com/incident/insert							
SOAP endpoint:	https://demo1.service-now.com/incident.do?SOAP							
Envelope:	<input type="button" value="XML"/> <input type="button" value="WSDL"/> <pre><SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP- ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tns="http://www.service- now.com/incident" xmlns:ms="http://www.service-now.com" xmlns:SOAP- ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP- ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"> <SOAP-ENV:Body> <insert xmlns="http://www.service-now.com"> <short_description xsi:type="xsd:string">\${short_description}</short_description> </insert> </SOAP-ENV:Body> </SOAP-ENV:Envelope></pre>							

Related Links

- [Preview script usage](#)
- [Test](#)

SOAP Message Parameters

New		SOAP Function = insert			1 to 1 of 1		
<input type="checkbox"/>	Name	Value					
<input type="checkbox"/>	short_description	this is the short description					
<input type="checkbox"/>	Actions on selected rows...						

The **SOAP action**, **SOAP endpoint**, and **Envelope** fields should be populated automatically based on the WSDL definition. The **Envelope** defines the message to send to the endpoint. In this example, the **Envelope** values have this format:

```
...
<!-- optional --><short_description xsi:type="xsd:string">
String</short_description>
...
```

To submit a specific value, enter the value directly in the appropriate XML tag. In this example, to set the Short description for a record, enter:

```
...
<short_description xsi:type="xsd:string">This is the short description</short_description>
...
```

To use variable substitution, use the format \${<variable_name>} instead of defining a specific value.

```
...
<short_description xsi:type="xsd:string">${short_desc}</short_description>
...
```

To test variable substitution after you have modified the SOAP envelope with the variables, define values for the variables in the SOAP Message Parameters related list. For example, click **New** and enter the following information:

Soap message parameters

SOAP Message Parameters		Update	Delete			
Name:	short_desc	SOAP Function:	insert			
Value:	this is the short descriptor					
Update Delete						

Test a SOAP message to validate the configuration before using the message in an integration.

To test the SOAP message, click the **Test** related link. You are redirected to a test result form as shown below.

Soap message test

The screenshot shows a web-based application for testing SOAP messages. At the top, there's a header with a back arrow, the title 'SOAP Message Test', and buttons for 'Update' and 'Delete'. Below the header, there are two main sections: 'Request' and 'Response'. The 'Request' section contains the XML code for a SOAP message to insert an incident. The 'Response' section shows the XML code for a successful response with an HTTP status of 200. At the bottom, there are 'Update' and 'Delete' buttons, and a 'Related Links' section with a 'Rerun test' link.

You can see the original SOAP request message, the resulting HTTP status code, and the SOAP response in this screen. You can also click the **Rerun test** related link to resubmit the SOAP request.

Note: A test SOAP message will time out after 60 seconds if a response is not received.

When creating SOAP message functions, you can configure the function to be sent through a MID Server.

There must be a running MID Server associated with your instance to use this functionality. All SOAP messages sent through a MID Server are performed asynchronously.

Soap message mid

The screenshot shows the 'SOAP Message Function' configuration screen. At the top, there are 'Update' and 'Delete' buttons. Below them, the 'Function' field contains 'TempConvertSoap.FahrenheitToCelsius'. The 'SOAP message' field is set to 'TemperatureConvert'. The 'Use MID server' field is populated with 'win2008server' and has a red border around it. There are several checkboxes: 'Lock' (checked), 'Use basic auth' (unchecked), 'Use WS-Security' (unchecked), and 'Strip whitespace' (unchecked). The 'SOAP action' field is 'http://tempuri.org/FahrenheitToCelsius' and the 'SOAP endpoint' field is 'http://www.w3schools.com/webservices/tempconvert.asmx'. The 'Envelope' section displays the following XML code:

```
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:me="http://tempuri.org/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
      <FahrenheitToCelsius xmlns="http://tempuri.org/">
        <!-- optional -->
        <Fahrenheit xsi:type="xsd:string">${temperatureF}</Fahrenheit>
      </FahrenheitToCelsius>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

By specifying a MID Server, all SOAP requests that use this SOAP message are sent through that MID Server. You can override the selected MID Server by using the `setMIDServer(mid server)` API call in a script.

Create a SOAP message from a WSDL and external XSD file.

Before you begin

Role required: web_service_admin

About this task

This task includes example WSDL and XSD files for a weather forecast SOAP message. Your WSDL and XSD file will vary.

Procedure

1. Navigate to **All > System Web Services > SOAP Message** and create a new record.
2. Clear the **Download WSDL** check box.
3. Paste the content of the WSDL into the **WSDL XML** field.

4. Save the record.
5. In the **SOAP Message Imports** related list, click **New**.
6. Paste the content of the XSD file into the **External Document** field.
7. Set the **Schema Location** field to db://<name of the referenced XSD file>.xsd.
Specifying the schema location allows the instance to know the location of the referenced XSD file.
8. Click **Submit**.
9. Click **Generate sample SOAP messages**.

Example: Example WSDL and XSD files

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://www.webservicex.net" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://www.webservicex.net" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get one week weather forecast for valid zip code or Place name in USA</wsdl:documentation>
    <wsdl:types>
        <s:schema elementFormDefault="qualified" targetNamespace="http://www.webservicex.net">
            <s:include schemaLocation="WeatherForecast.xsd" />
        </s:schema>
    </wsdl:types>
    <wsdl:message name="GetWeatherByZipCodeSoapIn">
        <wsdl:part name="parameters" element="tns:GetWeatherByZipCode" />
    </wsdl:message>
    <wsdl:message name="GetWeatherByZipCodeSoapOut">
        <wsdl:part name="parameters" element="tns:GetWeatherByZipCodeResponse" />
    </wsdl:message>
```

```
<wsdl:message name="GetWeatherByPlaceNameSoapIn">
    <wsdl:part name="parameters" element="tns:GetWeatherBy
PlaceName" />
</wsdl:message>
<wsdl:message name="GetWeatherByPlaceNameSoapOut">
    <wsdl:part name="parameters" element="tns:GetWeatherBy
PlaceNameResponse" />
</wsdl:message>
<wsdl:message name="GetWeatherByZipCodeHttpGetIn">
    <wsdl:part name="ZipCode" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherByZipCodeHttpGetOut">
    <wsdl:part name="Body" element="tns:WeatherForecasts"
/>
</wsdl:message>
<wsdl:message name="GetWeatherByPlaceNameHttpGetIn">
    <wsdl:part name="PlaceName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherByPlaceNameHttpGetOut">
    <wsdl:part name="Body" element="tns:WeatherForecasts"
/>
</wsdl:message>
<wsdl:message name="GetWeatherByZipCodeHttpPostIn">
    <wsdl:part name="ZipCode" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherByZipCodeHttpPostOut">
    <wsdl:part name="Body" element="tns:WeatherForecasts"
/>
</wsdl:message>
<wsdl:message name="GetWeatherByPlaceNameHttpPostIn">
    <wsdl:part name="PlaceName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherByPlaceNameHttpPostOut">
    <wsdl:part name="Body" element="tns:WeatherForecasts"
/>
</wsdl:message>
<wsdl:portType name="WeatherForecastSoap">
    <wsdl:operation name="GetWeatherByZipCode">
        <wsdl:documentation xmlns:wsdl="http://schemas.xmlso
ap.org/wsdl/">Get one week weather forecast for a valid Zi
p Code (USA) </wsdl:documentation>
        <wsdl:input message="tns:GetWeatherByZipCodeSoapIn"
/>
        <wsdl:output message="tns:GetWeatherByZipCodeSoapOut"
```

```
" />
    </wsdl:operation>
    <wsdl:operation name="GetWeatherByPlaceName">
        <wsdl:documentation xmlns:wsdl="http://schemas.xmlso
ap.org/wsdl/">Get one week weather forecast for a place n
ame (USA)</wsdl:documentation>
        <wsdl:input message="tns:GetWeatherByPlaceNameSoapIn
" />
        <wsdl:output message="tns:GetWeatherByPlaceNameSoapO
ut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="WeatherForecastHttpGet">
    <wsdl:operation name="GetWeatherByZipCode">
        <wsdl:documentation xmlns:wsdl="http://schemas.xmlso
ap.org/wsdl/">Get one week weather forecast for a valid Zi
p Code(USA)</wsdl:documentation>
        <wsdl:input message="tns:GetWeatherByZipCodeHttpGetI
n" />
        <wsdl:output message="tns:GetWeatherByZipCodeHttpGet
Out" />
    </wsdl:operation>
    <wsdl:operation name="GetWeatherByPlaceName">
        <wsdl:documentation xmlns:wsdl="http://schemas.xmlso
ap.org/wsdl/">Get one week weather forecast for a place n
ame (USA)</wsdl:documentation>
        <wsdl:input message="tns:GetWeatherByPlaceNameHttpGet
In" />
        <wsdl:output message="tns:GetWeatherByPlaceNameHttpGet
Out" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="WeatherForecastHttpPost">
    <wsdl:operation name="GetWeatherByZipCode">
        <wsdl:documentation xmlns:wsdl="http://schemas.xmlso
ap.org/wsdl/">Get one week weather forecast for a valid Zi
p Code(USA)</wsdl:documentation>
        <wsdl:input message="tns:GetWeatherByZipCodeHttpPostI
n" />
        <wsdl:output message="tns:GetWeatherByZipCodeHttpPost
Out" />
    </wsdl:operation>
    <wsdl:operation name="GetWeatherByPlaceName">
        <wsdl:documentation xmlns:wsdl="http://schemas.xmlso
```

```
ap.org/wsdl/">Get one week weather forecast for a place name (USA)</wsdl:documentation>
    <wsdl:input message="tns:GetWeatherByPlaceNameHttpPostIn" />
    <wsdl:output message="tns:GetWeatherByPlaceNameHttpPostOut" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="WeatherForecastSoap" type="tns:WeatherForecastSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetWeatherByZipCode">
        <soap:operation soapAction="http://www.webservicex.net/GetWeatherByZipCode" style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetWeatherByPlaceName">
        <soap:operation soapAction="http://www.webservicex.net/GetWeatherByPlaceName" style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="WeatherForecastSoap12" type="tns:WeatherForecastSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetWeatherByZipCode">
        <soap12:operation soapAction="http://www.webservicex.net/GetWeatherByZipCode" style="document" />
        <wsdl:input>
            <soap12:body use="literal" />
        </wsdl:input>
        <wsdl:output>
```

```
        <soap12:body use="literal" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetWeatherByPlaceName">
    <soap12:operation soapAction="http://www.webservicesex
.net/GetWeatherByPlaceName" style="document" />
    <wsdl:input>
        <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap12:body use="literal" />
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding name="WeatherForecastHttpGet" type="tns:We
atherForecastHttpGet">
    <http:binding verb="GET" />
    <wsdl:operation name="GetWeatherByZipCode">
        <http:operation location="/GetWeatherByZipCode" />
        <wsdl:input>
            <http:urlEncoded />
        </wsdl:input>
        <wsdl:output>
            <mime:mimeXml part="Body" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetWeatherByPlaceName">
        <http:operation location="/GetWeatherByPlaceName" />
        <wsdl:input>
            <http:urlEncoded />
        </wsdl:input>
        <wsdl:output>
            <mime:mimeXml part="Body" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="WeatherForecastHttpPost" type="tns:W
eatherForecastHttpPost">
    <http:binding verb="POST" />
    <wsdl:operation name="GetWeatherByZipCode">
        <http:operation location="/GetWeatherByZipCode" />
        <wsdl:input>
            <mime:content type="application/x-www-form-urlencoded" />
        </wsdl:input>
    </wsdl:operation>
</wsdl:binding>
```

```
</wsdl:input>
<wsdl:output>
    <mime:mimeXml part="Body" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetWeatherByPlaceName">
    <http:operation location="/GetWeatherByPlaceName" />
    <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
    </wsdl:input>
    <wsdl:output>
        <mime:mimeXml part="Body" />
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="WeatherForecast">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get one week weather forecast for valid zip code or Place name in USA</wsdl:documentation>
    <wsdl:port name="WeatherForecastSoap" binding="tns:WeatherForecastSoap">
        <soap:address location="http://www.webservicex.net/WeatherForecast.asmx" />
    </wsdl:port>
    <wsdl:port name="WeatherForecastSoap12" binding="tns:WeatherForecastSoap12">
        <soap12:address location="http://www.webservicex.net/WeatherForecast.asmx" />
    </wsdl:port>
    <wsdl:port name="WeatherForecastHttpGet" binding="tns:WeatherForecastHttpGet">
        <http:address location="http://www.webservicex.net/WeatherForecast.asmx" />
    </wsdl:port>
    <wsdl:port name="WeatherForecastHttpPost" binding="tns:WeatherForecastHttpPost">
        <http:address location="http://www.webservicex.net/WeatherForecast.asmx" />
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

```
<s:schema elementFormDefault="qualified" targetNamespace="http://www.webservicex.net" xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.webservicex.net" >
    <s:element name="GetWeatherByZipCode">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="ZipCode" type="s:string" />
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="GetWeatherByZipCodeResponse">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="GetWeatherByZipCodeResult" type="tns:WeatherForecasts" />
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:complexType name="WeatherForecasts">
        <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="Latitude" type="s:float" />
            <s:element minOccurs="1" maxOccurs="1" name="Longitude" type="s:float" />
            <s:element minOccurs="1" maxOccurs="1" name="AllocationFactor" type="s:float" />
            <s:element minOccurs="0" maxOccurs="1" name="FipCode" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="PlaceName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="StateCode" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Status" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="Details" type="tns:ArrayOfWeatherData" />
        </s:sequence>
    </s:complexType>
    <s:complexType name="ArrayOfWeatherData">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="unbounded" name="WeatherData" type="tns:WeatherData" />
        </s:sequence>
    </s:complexType>
</s:schema>
```

```
</s:sequence>
</s:complexType>
<s:complexType name="WeatherData">
    <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Day
" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Wea
therImage" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Max
TemperatureF" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Min
TemperatureF" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Max
TemperatureC" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Min
TemperatureC" type="s:string" />
    </s:sequence>
</s:complexType>
<s:element name="GetWeatherByPlaceName">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="P
laceName" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetWeatherByPlaceNameResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="G
etWeatherByPlaceNameResult" type="tns:WeatherForecasts" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="WeatherForecasts" type="tns:Weather
Forecasts" />
</s:schema>
```

Connectivity details

For the ServiceNow-initiated SOAP requests to successfully communicate with the web service provider inside a remote network, the ServiceNow

instance must have HTTP or HTTPS access to the SOAP endpoint at the provider.

Like any integration, such as LDAP, web services, or JDBC, the SOAP endpoint may reside behind a firewall that is blocking inbound communication from the instance. If this is the case, you need to make network changes to allow this connectivity into your network. You can either modify the firewall and ACL rules to allow the instance IP address, configure the SOAP message to [use a MID Server](#), or implement a [Virtual Private Network \(VPN\)](#) tunnel to allow the instance communication into your network.

Note: A common misconception is that because asynchronous SOAP requests are routed through the ECC queue, they are always sent through a MID Server. This is not the case. Asynchronous SOAP requests only use a MID Server when [configured to do so](#).

Outbound SOAP security

You can authenticate outbound SOAP messages using several different security protocols.

The security protocol you should use depends on the requirements of the web service provider. Mutual authentication is supported for outbound web services.

- [Enable basic authentication for outbound SOAP](#)

If the endpoint requires a user name and password, you can provide credentials using basic authentication.

- [Enable WS-Security for outbound SOAP](#)

You can sign outbound SOAP messages using username and password or a key store and trusted server certificate saved on the instance.

If the endpoint requires a user name and password, you can provide credentials using basic authentication.

Before you begin

Role required: web_service_admin

Procedure

1. Navigate to **All > System Web Services > Outbound > SOAP Message**.
2. Select a SOAP message record.
3. In the SOAP Message Functions related list, select a function.
4. Select **Use basic auth**.
5. Enter a user name in the Basic auth user ID field.
6. Enter the password for that user in Basic auth user password.
7. Click **Update**.

You can sign outbound SOAP messages using username and password or a key store and trusted server certificate saved on the instance.

Before you begin

Role required: admin

Procedure

1. Navigate to **All > System Web Services > Outbound > SOAP Message**.
2. Select a SOAP message record.
3. In the **SOAP Message Functions** related list, select a function.
4. In the **WS-Security type** field select the type of credentials to use, such as Username or X.509.
Some web service providers may require you to submit both types of credential.
5. In the **WS-Security x.509 profile** or **WS-Security Username profile** field, select the profile that contains the credentials you want to use.
If the record was configured with x.509 security prior to the Helsinki release, you can migrate the security settings to a WS-Security x.509 profile record by clicking the **Migrate to X509 Security Profile** button. When you click this button, security configuration field values (**Key store**, **Certificate**, **Key store alias**, and **Password**) are migrated to a new WS-Security x.509 profile record and that record is referenced

from the SOAP message function. If an x.509 profile record already exists with matching values, that record is referenced.

Note: Existing SOAP message functions that use the legacy security configuration will still work, however new records can use only an x.509 profile record for x.509 security.

6. Click **Update**.

Credentials from the selected profile are sent as part of the SOAP message header.

Related tasks

- [Create a new WS-Security profile](#)

Configure SOAP with a proxy

Certain properties provide support for SOAP requests to use a web proxy server.

SOAP properties

Property	Description	Example
glide.http.proxy_host	The proxy server hostname or IP address	proxy.company.com, 192.168.34.54
glide.http.proxy_port	The port number for the proxy server	8080, 9100
glide.http.proxy_username	If the proxy server is authenticating using user name and password, enter a value for this property	proxyuser
glide.http.proxy_password	If the proxy server is authenticating using user name and password	password

Property	Description	Example
	password, enter a value for this property	

Scripting outbound SOAP

You can send outbound SOAP requests from any place in the Now Platform where scripting is allowed.

For detailed API information about the server-side `SOAPMessageV2` and `SOAPResponseV2` APIs, see the API documentation on developer.servicenow.com. For additional tips and best practices, see the [Outbound REST Web Services `RESTMessageV2` and `SOAPMessageV2 execute\(\)` vs `executeAsync\(\)` Best Practices \[KB0694711\]](#) article in the Now Support Knowledge Base.

- [Preview a SOAP message script](#)

Generate a sample script to send the SOAP message.

- [Direct `SOAPMessageV2` example](#)

You can send an outbound SOAP message directly to the endpoint.

- [Asynchronous `SOAPMessageV2` example](#)

You can send an outbound SOAP message asynchronously.

- [SOAPMessageV2 MID server example](#)

You can send an outbound SOAP message through a MID Server.

- [Recordless `SOAPMessageV2` example](#)

You can use the `SOAPMessageV2()` constructor with no parameters to define a SOAP message entirely in the script.

Generate a sample script to send the SOAP message.

After you have developed and tested the SOAP message, click the **Preview script usage** related link in the SOAP Message Function form.

The dialog box displays an example of how you can invoke the SOAP message from a script.

You can manipulate the resulting XML response body with `XMLDocument` or with `gs.getXMLText` and `gs.getXMLNodeList`.

You can send an outbound SOAP message directly to the endpoint.

In this example, the script sends a SOAP message requesting a stock quote and waits for a response. If there is no response from the web service provider, or if the specified SOAP message record is unavailable, the script throws an error, handled in this example by the try-catch block.

```
var requestBody;
var responseBody;
var status;
var sm;
try{
    sm = new sn_ws.SOAPMessageV2("StockQuote", "GetQuote"); // Might throw exception if message doesn't exist or not visible due to scope
    sm.setBasicAuth("admin", "admin");
    sm.setStringParameter("symbol", "NOW");
    sm.setStringParameterNoEscape("xml_data", "<data>text</data>");
    sm.setHttpTimeout(10000) //In Milli seconds. Wait at most 10 seconds for response from http request.

    response = sm.execute(); //Might throw exception if http connection timed out or some issue with sending request itself because of encryption/decryption of password and stuff
    responseBody = response.haveError() ? response.getErrorMessage() : response.getBody();
    status = response.getStatusCode();
} catch(ex) {
    responseBody = ex.getMessage();
    status = '500';
} finally {
    requestBody = sm ? sm.getRequestBody():null;
}
gs.info("Request Body: " + requestBody);
```

```
gs.info("Response: " + responseBody);
gs.info("HTTP Status: " + status);
```

You can send an outbound SOAP message asynchronously.

When you send an asynchronous message the instance does not wait for a response before proceeding. You must handle waiting for a response within your code.

```
var requestBody;
var responseBody;
var status;
var sm;
try{
    sm = new sn_ws.SOAPMessageV2("StockQuote", "GetQuote"); // Might throw exception if message doesn't exist or not visible due to scope
    sm.setBasicAuth("admin","admin");
    sm.setStringParameter("symbol", "NOW");
    sm.setStringParameterNoEscape("xml_data","<data>test</data>");
    response = sm.executeAsync(); //Might throw exception if http connection timed out or some issue with sending request itself because of encryption/decryption of password

    response.waitForResponse(60); // In Seconds, Wait at most 60 seconds to get response from ECC Queue/Mid Server //Might throw exception timing out waiting for response in ECC queue

    responseBody = response.haveError() ? response.getErrorMessage() : response.getBody();
    status = response.getStatusCode();
} catch(ex) {
    responseBody = ex.getMessage();
    status = '500';
} finally {
    requestBody = sm ? sm.getRequestBody():null;
}
gs.info("Request Body: " + requestBody);
```

```
gs.info("Response: " + responseBody);
gs.info("HTTP Status: " + status);
```

You can send an outbound SOAP message through a MID Server.

By sending the message through a MID Server, you can access endpoints that are behind a firewall or within a private network. All SOAP messages sent through a MID Server are asynchronous.

```
var requestBody;
var responseBody;
var status;
var sm;
try{
    sm = new sn_ws.SOAPMessageV2("StockQuote", "GetQuote"); // Might throw exception if message doesn't exist or not visible due to scope
    sm.setBasicAuth("admin","admin");
    sm.setStringParameter("symbol", "NOW");
    sm.setStringParameterNoEscape("xml_data","<data>test</data>");
    sm.setMIDServer('mid_server_name');
    response = sm.execute(); //Might throw exception if http connection timed out or some issue with sending request itself because of encryption/decryption of password and stuff

    response.waitForResponse(60); // In Seconds, Wait at most 60 seconds to get response from ECC Queue/Mid Server //Might throw exception timing out waiting for response in ECC queue

    responseBody = response.haveError() ? response.getErrorMessage() : response.getBody();
    status = response.getStatusCode();
} catch(ex) {
    responseBody = ex.getMessage();
    status = '500';
} finally {
    requestBody = sm ? sm.getRequestBody():null;
}
gs.info("Request Body: " + requestBody);
```

```
gs.info("Response: " + responseBody);
gs.info("HTTP Status: " + status);
```

You can use the `SOAPMessageV2()` constructor with no parameters to define a SOAP message entirely in the script.

When using this constructor you must provide an endpoint and SOAP action. In this example, the script creates an empty SOAP message and sets the values needed to insert an incident record.

```
var s = new sn_ws.SOAPMessageV2(); //create an empty SOAP
message
s.setBasicAuth('admin','admin');
s.setSOAPAction('http://www.service-now.com/incident/inser
t'); //set the SOAP action to perform
s.setEndpoint('http://<instance>.service-now.com/incident.
do?SOAP'); //set the web service provider endpoint
s.setRequestBody('<soapenv:Envelope xmlns:soapenv=\\"http:/
/schemas.xmlsoap.org/soap/envelope\\" xmlns:inc=\\"http://w
ww.service-now.com/incident\\><soapenv:Header/><soapenv:Bo
dy><inc:insert><short_description>Test Dynamic SOAP</short
_description></inc:insert></soapenv:Body></soapenv:Envelop
e>');
var response = s.execute();
var xmlDoc = new XMLDocument(response.getBody());
var incident_sysid = xmlDoc.getNodeText('//sys_id');
```

Outbound web services: Logging

Log requests and responses for outbound web services such as REST and SOAP.

Outbound request logging allows you to better understand what 3rd-party services your instance accesses and the volume of outbound requests. Additionally, logging can provide valuable information when debugging outbound integrations.

Outbound web services logging tracks outbound REST and SOAP requests, as well as outbound requests made using the `GlideHTTPRequest` and `GlideHTTPClient` APIs.

To view outbound web service logs, navigate to **System Logs > Outbound HTTP Requests**.

All log information is read only.

- [Configure outbound logging](#)

You can configure outbound request logging to log basic, elevated, or all HTTP request and response information for specific domains.

- [Outbound request logging exclusion domain requirements](#)

When you exclude outbound request logging for a domain, the value you enter in the Domain field must meet certain requirements.

- [Outbound web service logging properties](#)

These properties allow you to control the behavior of outbound web service request logging.

Configure outbound logging

You can configure outbound request logging to log basic, elevated, or all HTTP request and response information for specific domains.

To configure the log level for a REST method or SOAP message function, navigate to the record you want to configure and click the **Set log level** related link, then select a log level for the current record.

To modify the log level for multiple outbound requests, navigate to **System Web Services > HTTP Log Levels** and change the log levels using the list. All outbound requests that have been configured with a specific log level are listed.

You can override the log level for all outbound requests using the properties `glide.outbound_http_log.override` and `glide.outbound_http_log.override.level`. Use these properties only for a limited time when troubleshooting.

You can set the log level in a script using the `setLogLevel()` function from the `SOAPMessageV2` and `RESTMessageV2` APIs. For more information about using these APIs, refer to the API documentation.

Related concepts

- [RESTMessageV2 - setLogLevel\(\)](#)

- SOAPMessageV2 - setLogLevel()

Outbound request log levels

Certain elements are logged based on the configured log level.

Logged elements

The following elements from the request and response are logged depending on the configured log level.

Logged elements by level

Field	Basic	Elevated	All
Sequence	x	x	x
HTTP Method	x	x	x
Protocol	x	x	x
Scheme	x	x	x
Hostname	x	x	x
Path	x	x	x
HTTP response status	x	x	x
Request length	x	x	x
Response length	x	x	x
Total call time	x	x	x
System ID	x	x	x
Source table	x	x	x
Source record ID	x	x	x

Field	Basic	Elevated	All
Note: You must have read access for records in the specified Source table to view this field.			
Scope	x	x	x
Session	x	x	x
Transaction	x	x	x
User	x	x	x
MID server	x	x	x
Request headers		x	x
Request query		x	x
Response headers		x	x
Request body			x
Response body			x

Outbound request logging exclusion domain requirements

When you exclude outbound request logging for a domain, the value you enter in the Domain field must meet certain requirements.

Requirements

The value you enter in the **Domain** field must meet the following requirements.

- Begins with HTTP:// or HTTPS://.
- Is a domain pattern or IP address.
- Ends with alphanumeric characters preceded by a period, such as .com.
- Includes at most a single wildcard character immediately following the scheme and hierarchical portion of the domain pattern.

Wildcard

You can use a single wildcard character (*) in the domain pattern. Use this wildcard immediately following the scheme and hierarchical portion of the domain pattern, such as http://*.domain.com to include all subdomains. The wildcard must immediately follow the scheme and hierarchical portion of the domain pattern. If you use an IP address instead of domain pattern, you must enter the full IP address without a wildcard.

Note: You cannot use multiple wildcards, or specify a wildcard without a domain pattern. Values such as * or *.* are not supported.

Domain matching

When evaluating the Origin header in a request, ServiceNow prioritizes rules that match the domain pattern exactly. If no exact match is found, the next closest match is used.

For example, if there are rules for the domain patterns http://*.blog.mysite.com and http://*.mysite.com, a request from http://alice.blog.mysite.com will match the http://*.blog.mysite.com pattern.

Examples of valid and invalid domains

Examples

Valid domain	Invalid domain
http://*.ms.net	https://*com
https://*.ms.com	http://*..com
https://*.com.au	http://192.168.1.*
http://192.168.1.1	http://*.168.1.126
http://*.service-now.com	http://blog.*.service-now.com
http://*.com	http://*com

Outbound request logging exclusion

You can exclude domains to allow only basic-level logging on those domains.

To exclude a domain, navigate to **System Web Services > Outbound HTTP Log Domain Exclusion List** and create a new record. You must have the `web_service_admin` role to exclude domains.

Outbound web service logging properties

These properties allow you to control the behavior of outbound web service request logging.

Property	Description
<code>glide.outbound_http.db.log</code>	By default, log information is written to a log file and to the instance database. Set this property to false to disable database logging. <ul style="list-style-type: none">• Type: true false

Property	Description
	<ul style="list-style-type: none">• Default value: true• Location: Add to the System Property [sys_properties] table
glide.outbound_http.text.content_types	<p>A comma-separated list of content types. The body of requests or responses with one of these content types is logged. By default, the content types text/*, application/json, and application/xml are always logged. Use this property to add additional content types.</p> <ul style="list-style-type: none">• Type: String• Default value: none• Location: Add to the System Property [sys_properties] table
glide.outbound_http.content.max_limit	<p>The maximum number of characters logged from a request or response body. The first characters of the body, up to this limit, are logged. This property has a maximum possible value of 1000.</p> <ul style="list-style-type: none">• Type: integer• Default value: 100• Location: Add to the System Property [sys_properties] table
glide.outbound_http_log.override	When this property is set to true, the property glide.outbound_http_log.override.level determines the log level for all

Property	Description
	<p>requests and responses. This log level overrides any other log level settings. Only use this property for a limited time when troubleshooting.</p> <ul style="list-style-type: none">• Type: true false• Default value: false• Location: Add to the System Property [sys_properties] table
glide.outbound_http_log.override.level	<p>The log level to use for all requests and responses when glide.outbound_http_log.override is true. Valid values are basic, elevated, and all.</p> <ul style="list-style-type: none">• Type: String• Default value: basic• Location: Add to the System Property [sys_properties] table

Additional integration resources

In addition to employing inbound and outbound web services to integrate with ServiceNow apps, other resources are available to you, such as scripted GraphQL, Javascript, and API developer guides.

- [Using extension points to extend application functionality](#)

Use extension points to extend the functionality of an application without altering the original application code. You can use pre-existing extension points available in selected Now Platform applications, or you can add extension points when you develop custom applications in your own instance.

- [Manage connection pooling](#)

Outbound HTTP(S) connections from a base system instance or inbound connections from MID Servers, the ODBC driver, and other clients are maintained and reused where possible.

- [Create data sources from other apps using ODBC driver](#)

The ServiceNow open database connectivity (ODBC) driver provides read-only access to the database associated with your ServiceNow instance.

- [Understanding domain separation and web services](#)

Domain separation is supported in Web Services. Domain separation enables you to separate data, processes, and administrative tasks into logical groupings called domains. You can control several aspects of this separation, including which users can see and access data.

- [Analyze REST and SOAP API usage](#)

REST and SOAP API analytics allow you to track and analyze web service API usage.

- [Query record data using the GraphQL API framework](#)

Create a custom GraphQL API to query record data from a component or a third-party system.

Using extension points to extend application functionality

Use extension points to extend the functionality of an application without altering the original application code. You can use pre-existing extension points available in selected Now Platform applications, or you can add extension points when you develop custom applications in your own instance.

Using extension points

By using extension points, you can integrate customizations without actually altering the core components in the application code. Extension points can prevent your custom code interactions from breaking, which often occurs after an upgrade if you directly embed the custom code into the application code.

Extension points that are embedded in the application code act as out-points, where data passes to the custom code, and as in-points that handle the returned results. When creating an application, the returned data or objects must conform to the requirements that you define for the extension point.

Note: Some out-of-box configured CSM Query Rules cannot be changed or disabled because they are also used in constants within a business rule.

Types of extension points

You can create extension points to process the custom code that uses the following types of artifacts:

Scripted extension points

Extension points in server-side script includes that store JavaScript functions and object classes. To learn about scripts includes and how to implement scripted extension points in the application and custom code, see:

- [Script includes](#)
- [Registering custom script includes against the scripted extension points](#)

UI extension points

Extension points that are used in server-side UI macros such as HTML extensions.

UI macros are discrete scripted components that you can add to the user interface. You use them to add custom content to a UI page, without having to directly modify the page. For example, you can use UI macros to add headers and footers to the standard Knowledge Base View (kb_view) UI page in which KB articles appear.

To learn about UI macros and how to implement UI extension points in the application and custom code, see:

- [UI Macros](#)
- [Creating and adding a UI extension point](#)

- Registering custom script includes against the scripted extension points

Client extension points

Extension points that are used in client-side UI scripting, typically for modifying forms.

UI scripts enable you to package client-side JavaScript into a reusable form, which is similar to how script includes store server-side JavaScript. You can create UI scripts and run them from client scripts, from other client-side script objects, and from HTML code.

To learn about UI scripts and how to implement client extension points in application and custom code, see:

- UI scripts
- Creating and adding a client extension point
- Registering custom script includes against the scripted extension points
Registering custom script includes against the scripted extension points

When you use extension points to process customizations, you create a defined structure for integrating custom data or functionality into an application. Custom server-side script includes, UI macros such as HTML extensions, and client-side UI scripts are all external to the application code and only interact with it at specified extension points.

Application code

The term application code refers to:

Standard application code

Standard, or base, application code that comprises the Now Platform. Pre-defined extension points are already embedded in certain applications, such as Customer Service Management and Field Service Management. To learn more about the Now Platform applications that contain pre-defined extension points, see the following:

Application	Extension point topic
Coaching	Coaching troubleshooting

Application	Extension point topic
Continual Improvement Management	Improvement with other applications
Customer Service Management	Extension points in Customer Service Management
	Creating custom user roles
	CSM integration with Change Management
	CSM integration with Incident Management
	CSM integration with Problem Management
	CSM integration with Request Management
Field Service Management	Extension points in Field Service Management
Knowledge Management	Use extension points for Knowledge Management
Orchestration	Client software distribution extension network Installed with client software distribution
Password Reset	Password Reset script includes

Internally developed custom applications for your enterprise

You can add extension points to handle the registration of custom artifacts that are used to modify or extend the functionality of an application. When developers create custom code, they register, or pair, specific custom artifacts with specific extension points. Adding extension points enables integration of future customizations without having to change your base code.

Creating an extension point in the application code

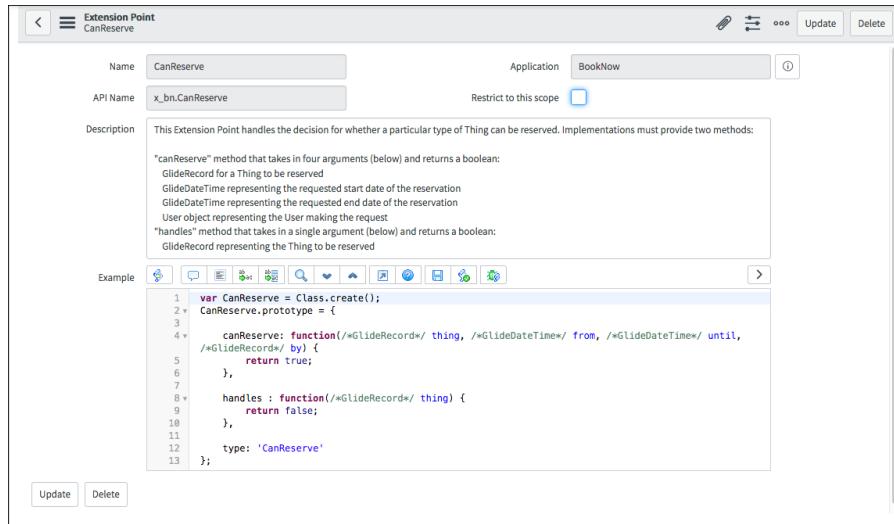
You want to ensure that the internally developed applications that you create for your enterprise can be properly customized, combined, and upgraded as needed. Extension points enable you to modify the functionality and user interface for an application without editing its core components. Use of extension points also creates a highly defined structure for functionality extensions.

When you create an extension point, you can restrict its use to the application scope in which it is defined or specify that the extension point can run in all scopes globally. If the application scope is restricted, customizations that are registered against the extension point can only run in the designated application scope. Before designing and building an application that includes extension points, you should:

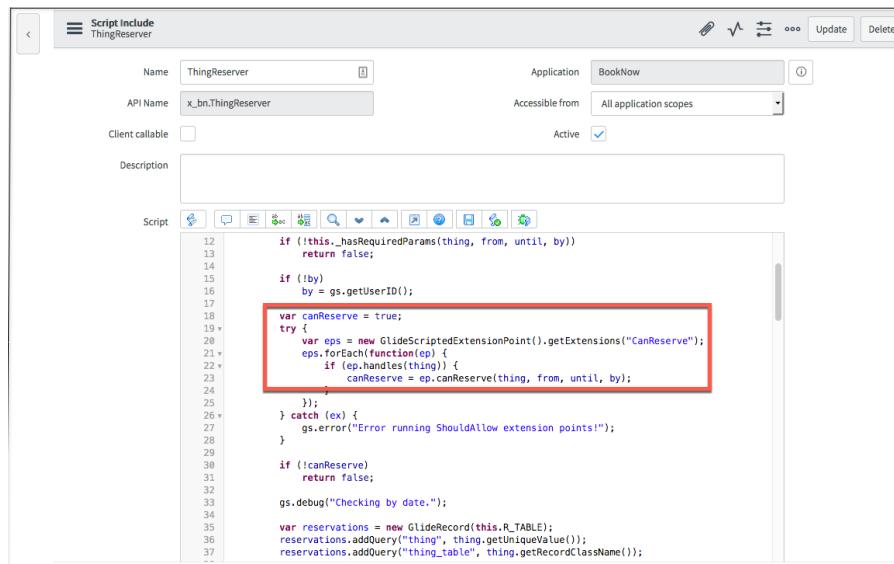
- Identify where to place extension points in the application code to accommodate custom script includes, UI macros, or UI scripts.
- Determine the content and structure for an extension point. This definition should describe how a customization should be structured, how it interacts with the application code, and how the data that is returned by a customization at the extension point is processed.

To create extension points and embed them in the application code, perform the following tasks:

1. Create an extension point and define its custom interface. This is an example of a scripted extension point.



2. Designate specific places in the application code where data or objects can be sent to a customization, and where data is returned.
3. Create an artifact, such as a script include, UI macro, or UI script, that calls the [GlideScriptedExtensionPoint API - Scoped](#) method. This method identifies the extension point at which registered custom artifacts execute in the application code.



Enabling debugging and logging

When you create a extension point, you should also enable debugging and logging. Debugging and logging help you to identify and fix issues that are related to the use of custom artifacts with an extension point. When you enable the debugger, you can set break points in script includes, UI macros, or UI scripts. You can check the logs to see the following details:

- When an extension point finds active extension instances, including the total number of extension instances found and the time each extension instance was found.
- When an extension point executes extension instances, including the total number of custom artifacts executed and the time of each executed. It also includes the total number that failed to execute and the time at which each extension instance failed.

Note: To learn more about how to enable debugging and how it works, see:

- [Script Debugger and Session Log](#)
- [Debugging applications](#)
- [Writing to the debug log](#)

Registering a custom artifact against an extension point

When you are customizing an application, you extend the base functionality by using custom artifacts, such as server-side script includes, UI macros, or client-side UI scripts. To design and build custom artifacts, perform the following tasks:

1. Review the listing of available extension points that are appropriate to the specific type of custom artifact that you are creating.
 - These listings include extension points that you created and any pre-defined extension points that are embedded in the Now Platform functions.

- Each listing includes information about an extension point in the application code that calls a custom artifact and what data or objects should be returned to it.
2. Select an extension point.
 3. Determine how to structure the custom artifact. The structure should be based on the extension point descriptions. The descriptions include the requirements for using the artifact with the custom code and where the artifact will located in the base application code.
 4. Create the custom artifact and code when you register it against the selected extension point. Through registration, you create an extension instance record that links the extension point definition to its implementation in the custom artifact.

How registered custom artifacts are processed

When the application code executes and finds an embedded API call containing an extension point, it:

1. Uses the extension point in the API call to determine which custom artifacts are registered against it.
 2. Sends the appropriate data or objects to the registered custom artifacts.
 3. Collects the returned output from each custom artifact.
 4. Processes and incorporates the returned results into the base application.
- [Using scripted extension points in server-side scripts](#)

Use the scripted extension points in the server-side script includes that store JavaScript functions and object classes. By using extension points, you can integrate customizations without actually altering the core components in the application code.

- [Using UI extension points in server-side UI macros](#)

Use UI extension points in the server-side UI macros, such as HTML extensions, to add custom content to a UI page without having to directly modify the page. By using UI extension points, you can

integrate customizations without actually altering the core components in the application code.

- [Using client extension points in client-side UI scripting](#)

Use client extension points in client-side UI scripting to modify forms, so that you do not have to directly modify the form. By using client extension points, you can integrate customizations without actually altering the core components in the application code.

Using scripted extension points in server-side scripts

Use the scripted extension points in the server-side script includes that store JavaScript functions and object classes. By using extension points, you can integrate customizations without actually altering the core components in the application code.

You create the scripted extension points and add them to the script includes in the base application code. When customizing a base application, you implement the scripted extension points by creating the custom script includes and registering them against the scripted extension points.

Note: For an example of a scripted extension point and to learn more about this process, see [Using extension points to extend application functionality](#).

- [Creating and adding a scripted extension point](#)

When developing an application, create scripted extension points and add them to the script includes in the application code. Use a scripted extension point to designate the specific location where data or objects can be sent to a registered custom script include and where returned results are processed.

When developing an application, create scripted extension points and add them to the script includes in the application code. Use a scripted extension point to designate the specific location where data or objects can be sent to a registered custom script include and where returned results are processed.

Note: For an example of a scripted extension point, see [Using extension points to extend application functionality](#).

- [Create a scripted extension point](#)

Create a scripted extension point that can be placed in a script include in the base application code. By placing the scripted extension point, you designate the specific location in the application code where data or objects can be sent to a customization and where data is returned.

- [Add a scripted extension point in the base application code](#)

Add the scripted extension point into a script include in the base application code. To add the extension point, include an API call that identifies the location at which registered custom artifacts execute.

Create a scripted extension point that can be placed in a script include in the base application code. By placing the scripted extension point, you designate the specific location in the application code where data or objects can be sent to a customization and where data is returned.

Before you begin

Role required: You must have the specific role for the developer or administrator of the application, or you must have the admin role.

Note: To learn about application-specific administrator roles and delegated development, see [Access control rules in application administration apps](#) and [Delegated development and deployment](#).

About this task

Define the content and structure for an extension point. This definition should describe how a customization should be structured, how it interacts with the application code, and how the data that is returned by a customization at the extension point is processed.

By creating a good definition, you provide a structure for the extensions.

Procedure

1. Navigate to **All > System Extension Points > Scripted Extension Points**.
2. Click **New**.
3. On the form, fill in the fields.

Scripted Extension Point form

Field	Description
Name	Unique name for the extension point. The name can be up to 100 alphanumeric characters, including special characters.
API Name	Name of the extension point API that is pre-pended with the application scope to which it applies. This is a system-assigned name and cannot be changed.
Application	Application scope against which the extension point is assigned. This is system-assigned and cannot be changed. For more information about the protections that are offered by the use of scoping, see Application scope .
Restrict to this scope	Option for restricting the extension point to the application scope only.
Description	Requirements for the custom script include, such as how the UI script should be structured and how it should operate with the application code.
Example	Example of how a custom script include that interacts with

Field	Description
	this extension point should be structured so it operates with the application code.

4. Click **Submit**.

Add the scripted extension point into a script include in the base application code. To add the extension point, include an API call that identifies the location at which registered custom artifacts execute.

Before you begin

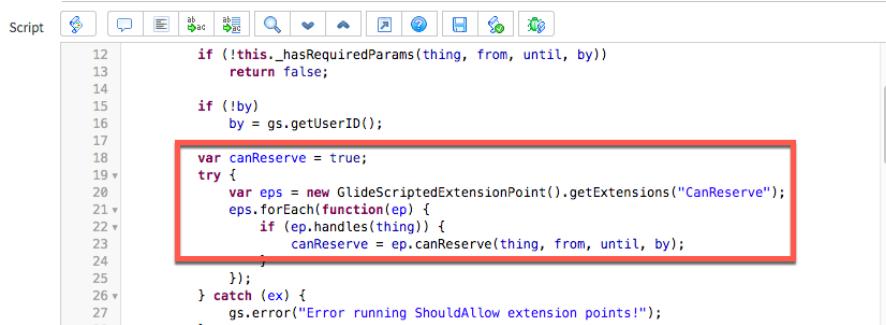
Role required: You must have the specific role for the developer or administrator of the application, or you must have the admin role. You must first create a scripted extension point before you can add it to a script include in the application code.

Note: To learn about application-specific administrator roles and delegated development, see [Access control rules in application administration apps](#) and [Delegated development and deployment](#).

Procedure

1. In the application code, access the existing script include that you want to add a scripted extension point to, or create a new script include.
To learn more about creating script includes, see [Script includes](#).
2. In the script include, add a line of code at the location that you expect to collect and process the custom script output.

The code must contain the [GlideScriptedExtensionPoint API - Scoped](#) method that identifies the extension point against which the custom script includes are registered. For example:



```
Script
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
--  
if (!this._hasRequiredParams(thing, from, until, by))  
    return false;  
  
if (!by)  
    by = gs.getUserID();  
  
var canReserve = true;  
try {  
    var eps = new GlideScriptedExtensionPoint().getExtensions("CanReserve");  
    eps.forEach(function(ep) {  
        if (ep.handles(thing)) {  
            canReserve = ep.canReserve(thing, from, until, by);  
        }  
    });  
} catch (ex) {  
    gs.error("Error running ShouldAllow extension points!");  
}
```

Note: It is also a good practice to deliver error messages for application customizers when their custom scripts run in an extension instance. If something fails when the function returns arguments to the extension point, the error messages include information about how to troubleshoot the custom script.

Using UI extension points in server-side UI macros

Use UI extension points in the server-side UI macros, such as HTML extensions, to add custom content to a UI page without having to directly modify the page. By using UI extension points, you can integrate customizations without actually altering the core components in the application code.

You create the UI extension points and add them to the UI macros in the base application code. When customizing a base application, implement the UI extension points by creating the custom UI macros and registering them against the UI extension points.

Note: You create and register the UI extension points in a manner that is similar to how you implement the scripted extension points. For a detailed example of a scripted extension point, and to learn more about this process, see [Using extension points to extend application functionality](#).

- Creating and adding a UI extension point

When developing an application, create UI extension points and add them to the server-side UI macros, such as HTML extensions, or the UI pages in the base application code.

- Create a UI extension point

Create a UI extension point that you can place in the UI macros, such as HTML extensions, in the application code. By placing the UI extension point, you designate the specific location in the application code where data or objects can be sent to a customization and where data is returned.

- [Add a UI extension point in the base application code](#)

Add the UI extension point into a UI macro or HTML extension in the base application code. To add the extension point, place an API call that identifies the location at which registered custom artifacts execute.

When developing an application, create UI extension points and add them to the server-side UI macros, such as HTML extensions, or the UI pages in the base application code.

Use a UI extension point to designate the specific location where data or objects can be sent to a registered custom UI macro and where returned results are processed.

Create a UI extension point that you can place in the UI macros, such as HTML extensions, in the application code. By placing the UI extension point, you designate the specific location in the application code where data or objects can be sent to a customization and where data is returned.

Before you begin

Role required: admin or the specific role for the developer or administrator of the application

Note: To learn about application-specific administrator roles and delegated development, see [Access control rules in application administration apps](#) and [Delegated development and deployment](#).

About this task

Define the content and structure for an extension point. This definition should describe how a customization should be structured, how it interacts with the application code, and how the data that is returned by a customization at the extension point is processed.

By creating a good definition, you provide a structure for the functionality of the extensions.

Procedure

1. Navigate to **All > System Extension Points > UI Extension Points**.
2. Click **New**.
3. On the form, fill in the fields.

UI Extension Point form

Field	Description
Name	Unique name for the extension point. The name can be up to 100 alphanumeric characters, including special characters.
API Name	Name of the extension point API that is pre-pended with the application scope to which it applies. This is a system-assigned name and cannot be changed.
Application	Application scope against which the extension point is assigned. This is system-assigned and cannot be changed. For more information about the protections that are offered by the use of scoping, see Application scope .
Restrict to this scope	Option for restricting the extension point to the application scope only.
Allow access over AJAX/REST	Option for enabling or disabling access to the UI extension point over AJAX or REST.

Field	Description
	To learn more about these development architectures and techniques, see AJAX and REST APIs .
Description	Requirements for the custom UI macro, such as how the UI script should be structured and how it should operate with the application code.
Example	Example of how a custom UI macro that interacts with this extension point should be structured so it operates with the application code.

4. Click **Submit**.

Add the UI extension point into a UI macro or HTML extension in the base application code. To add the extension point, place an API call that identifies the location at which registered custom artifacts execute.

Before you begin

You must first create a UI extension point before you can add it to a UI macro or UI page in the application code. Role required: admin or the specific role for the developer or administrator of the application

Note: To learn about application-specific administrator roles and delegated development, see [Access control rules in application administration apps](#) and [Delegated development and deployment](#).

Procedure

1. In the application code, access the existing UI macro that you want to add a scripted extension point to or create a new one.
To learn more about creating UI macros with jelly tags, see [UI Macros](#) and [Jelly tags](#).

2. In the UI macro, add a jelly tag with a line of code at the location that you expect to collect and process custom UI macro output. Typically, this line of code contains:

- A `call_extension` command that identifies the name of the UI extension point (for example, `extension="global.KMArticle.ViewHeader"`) against which the custom UI macros are registered.

This line of code is similar to what is in an application code script include for scripted extension points. For an example, see [Register a custom script include](#).

- Arguments that are passed into the UI macro when it is rendered. For example, for a Knowledge Base article, `knowledgerecord="${knowledgeRecord}"` passes in the current knowledge record.
- (Optional) If there are multiple implementations of this extension point, using a `limit` command specifies that `x` number of implementations should be returned (for example, `limit="1"`). This figure shows a UI macro that contains these commands.

The screenshot shows the ServiceNow Macro editor interface. At the top, there's a toolbar with icons for New, Edit, Delete, and Save. Below the toolbar, the macro details are shown: Name (kb_view_common), Application (Global), and Active (checked). The Description field is empty. The XML tab is selected, displaying the following code:

```
<!-- KB Extension point call to customize footer -->
<@call_extension extension="global.KBViewArticleViewHeader" knowledgeRecord="${knowledgeRecord}" limit="1" />
```

A red box highlights the line above, indicating it is a customization point. The code continues with conditional logic for mobile and non-mobile views.

At the bottom left, there's a note about PDF generation and copyright information. On the right side, the page number 334 is visible.

PDF generated on July 15, 2023
©2023 ServiceNow. All rights reserved. [Terms of Use](#) [Privacy Statement](#)

334

Using client extension points in client-side UI scripting

Use client extension points in client-side UI scripting to modify forms, so that you do not have to directly modify the form. By using client extension points, you can integrate customizations without actually altering the core components in the application code.

You create client extension points and add them to the UI scripts in the base application code. When customizing a base application, implement the client extension points by creating the custom UI scripts and registering them against the client extension points.

Note: You create and register the client extension points in a manner that is similar to how you implement the scripted and UI extension points. For a detailed example of a scripted extension point, and to learn more about this process, see [Using extension points to extend application functionality](#) and [Using UI extension points in server-side UI macros](#).

- [Creating and adding a client extension point](#)

Create client extension points and add them to the client-side UI scripts in the base application code.

Create client extension points and add them to the client-side UI scripts in the base application code.

Use a client extension point to designate the specific location where data or objects can be sent to a registered custom UI script and where returned results are processed.

- [Create a client extension point](#)

Create a client extension point that you can place in the UI scripts in the application code. By placing the client extension point, you designate the specific location in the application code where data or objects can be sent to a customization, and where data is returned.

- [Add a client extension point in the base application code](#)

Add the UI extension point into a UI script in the base application code. To add the extension point, place an API call that identifies the location at which registered custom artifacts execute.

Create a client extension point that you can place in the UI scripts in the application code. By placing the client extension point, you designate the specific location in the application code where data or objects can be sent to a customization, and where data is returned.

Before you begin

Role required: admin or the specific role for the developer or administrator of the application

Note: To learn about application-specific administrator roles and delegated development, see [Access control rules in application administration apps](#) and [Delegated development and deployment](#).

About this task

Define the content and structure for an extension point. This definition should describe how a customization should be structured, how it interacts with the application code, and how the data that is returned by a customization at the extension point is processed.

By creating a good definition, you provide a structure for your extension points.

Procedure

1. Navigate to **All > System Extension Points > Client Extension Points**.
2. Click **New**.
3. On the form, fill in the fields.

Client Extension Point form

Field	Description
Name	Unique name for the extension point. The name can be up

Field	Description
	to 100 alphanumeric characters, including special characters.
API Name	Name of the extension point API that is pre-pended with the application scope to which it applies. This is a system-assigned name and cannot be changed.
Application	Application scope against which the extension point is assigned. This is system-assigned and cannot be changed. For more information about the protections that are offered by the use of scoping, see Application scope .
Restrict to this scope	Option for restricting the extension point to the application scope only.
Allow access over AJAX/REST	Option for enabling access to the client extension point over AJAX or REST. To learn more about these development architectures and techniques, see AJAX and REST APIs .
Description	Requirements for the custom UI script, such as how the UI script should be structured and how it should operate with the application code.
Example	Example of how a custom UI script that interacts with this extension point should be

Field	Description
	structured so it operates with the application code.

4. Click **Submit**.

Add the UI extension point into a UI script in the base application code. To add the extension point, place an API call that identifies the location at which registered custom artifacts execute.

Before you begin

You must first create a UI extension point before you can add it to a UI script in the application code. Role required: admin or the specific role for the developer or administrator of the application

Note: To learn about application-specific administrator roles and delegated development, see [Access control rules in application administration apps](#) and [Delegated development and deployment](#).

Procedure

1. In the application code, access the existing UI script that you want to add a scripted extension point to, or create a new one. To learn more about creating UI scripts, see [UI scripts](#).
2. In the UI script, add a line of code at the location that you expect to collect and process the custom UI script output.

The code must contain the `getExtensions` command that identifies the client extension point against which the custom UI scripts are registered. This line of code is similar to what is in an application code script include for scripted extension points. For an example, see [Register a custom script include](#).

Manage connection pooling

Outbound HTTP(S) connections from a base system instance or inbound connections from MID Servers, the ODBC driver, and other clients are maintained and reused where possible.

Connection pooling is used to keep track of HTTP(S) client connections to determine if they are alive and available for reuse.

ServiceNow HTTP client code means:

- Any application or script which makes outbound HTTP(S) requests from a base system instance.
- ServiceNow code in the MID Server or the ODBC driver which makes HTTP(S) requests to one or more base system instances.

Note: This discussion does not apply to browser-to-instance communication. No changes have been made with respect to the management of HTTP(S) connections for browser-based communication with ServiceNow. This discussion also does not apply to customer-developed Web Services clients making requests to ServiceNow.

What Should the Customer Do?

Users should monitor performance, such as the decreased time for loading Discovery data and improved ODBC driver performance. For systems with an unusually large amount of simultaneous outbound HTTP(S) activity, such as numerous third-party integrations or high-volume automated activities which generate HTTP(S) requests from the base system instance to other places, review the `max_connections` and `max_connections_per_host` properties to ensure that the settings are sufficient. This enhancement has no impact on end-user connections from browsers and no impact on connections from customer-developed Web Services client applications.

- [HTTP Connection Management Properties](#)

Connection pooling is controlled by three properties.

HTTP Connection Management Properties

Connection pooling is controlled by three properties.

The default values for these properties are appropriate for most customers. The Glide properties are dynamic, meaning that changes to these properties will take effect immediately. No outage or restart is required to update the values.

Property	Description	Default Value
glide.http.connection_mgr.max_connections	Controls the total number of permitted HTTP(S) connections outbound from the base system instance. This is an instance-wide value.	20
glide.http.connection_mgr.max_connections_per_host	Controls how many of the glide.http.connection_mgr.max_connections can communicate in parallel with any particular host. If the maximum setting for any of these values is reached during normal operations, a script or background thread may have to wait briefly to obtain a connection.	4
glide.http.outbound.max_timeout	The max timeout in seconds to wait for SOAPMessageV2 and RESTMessageV2 when using executeAsync method. The maximum value allowed is 30 seconds. If this value is set to something higher than 30 seconds, it will default to 30 seconds.	30

Property	Description	Default Value
glide.http.outbound.max_timeout.enabled	Determines whether a configurable hard maximum wait time is enforced for SOAPMessageV2 and RESTMessageV2 when using executeAsync method.	true
glide.http.use_connection_mgr	Switches connection pooling on and off. To disable the new behavior (not recommended) set glide.http.use_connection_mgr to false.	true

Create data sources from other apps using ODBC driver

The ServiceNow open database connectivity (ODBC) driver provides read-only access to the database associated with your ServiceNow instance.

Note: Versions of the ODBC driver older than 1.0.7.3 are no longer supported.

The ODBC driver is compliant to version 3.52 of the Microsoft ODBC core API conformance. The ServiceNow ODBC driver uses ServiceNow web services support for a query-only interface. The ODBC driver supports only select statements or read-only functions and does not modify your instance data. Because the ODBC driver uses the web services interface, platform-wide access control (ACL) is enforced and data security is in place.

Note:

The ODBC driver has these limitations:

- The ODBC driver supports only select statements or read-only functions, and does not modify your instance data.
- There is no supported way to use the ODBC driver with a Java client application or with a Java JDBC-ODBC bridge.
- There is a hard-coded limit of 512 characters when accessing views through ODBC. Because of this limitation, a maximum of 11 table sys_ids can be included in an ODBC view query. Anything over 11 tables results in an error.

- [Getting started with ODBC](#)

Before installing the ODBC driver, view the video on this page, create an ODBC user account, assign the odbc role, and define an ACL rule for the odbc role.

- [Installing the ODBC driver](#)

Review setup requirements, download the ODBC driver installer, and install the ODBC driver to a computer.

- [Configuring the ODBC driver](#)

After installing the ODBC driver, configure it to connect to your ServiceNow instance and to communicate through a proxy server, if applicable, and set properties to control ODBC behavior.

- [Test the ODBC driver](#)

After configuring the ODBC driver, test that the driver can connect to the base instance as the ODBC user and can query data from a target table.

- [ODBC behavior](#)

After testing the ODBC driver you can use it to query your instance database from a variety of client applications.

- [ODBC and client applications](#)

See the following pages for examples of how to use the ODBC driver to create data sources from other applications.

- [Domain separation and ODBC driver](#)

This is an overview of domain separation and ODBC drivers. Domain separation enables you to separate data, processes, and administrative tasks into logical groupings called domains. You can control several aspects of this separation, including which users can see and access data.

Getting started with ODBC

Before installing the ODBC driver, view the video on this page, create an ODBC user account, assign the odbc role, and define an ACL rule for the odbc role.

A ServiceNow user with the admin role can perform these procedures.

Watch this seven-minute video for an overview of installing, configuring and testing the ServiceNow [Create data sources from other apps using ODBC driver](#), which provides read-only access to the database associated with your ServiceNow instance.

Before downloading and installing the ODBC driver, [review the requirements](#) to ensure that your configuration is compatible.

1. [Create an ODBC user account and assign the odbc role](#)

The ODBC driver communicates with your ServiceNow instance as a specific user. Create an ODBC user account and assign the odbc role to enable the user to communicate via ODBC.

2. [Define an ACL rule for the odbc role](#)

Define an ACL rule for the odbc role to provide read access to the incident table. You can create other ACL rules for the odbc role to provide read access to other tables.

The ODBC driver communicates with your ServiceNow instance as a specific user. Create an ODBC user account and assign the odbc role to enable the user to communicate via ODBC.

Before you begin

Role required: admin

About this task

The odbc role contains various additional roles, including the soap_query role required to make ODBC requests and the itil role required to access core tables such as the incident table. You can modify the odbc role to allow access to other tables as needed by adding additional child roles.

Procedure

1. Navigate to **All > User Administration > Users**.
2. Click **New**.
3. In the **User ID** field, enter `odbc.user`.
4. Enter a **Password** for this user.
5. From the form context menu, select **Save**.
6. In the **Roles** related list, click **Edit**.
7. Use the slushbucket to add the odbc role, and then click **Save**.
8. Click **Submit**.

What to do next

In a separate browser session, confirm that the `odbc.user` is able to log in to your ServiceNow instance.

Define an ACL rule for the odbc role to provide read access to the incident table. You can create other ACL rules for the odbc role to provide read access to other tables.

Before you begin

Role required: admin

Procedure

1. Elevate the session permissions to security_admin so you can create ACL rules.
2. Navigate to **System Security > Access Controls (ACL)**.
3. Click **New**.
4. From the **Operation** choice list, select **read**.
5. From the **Name** choice list, select **Incident [incident]**.

Leave the second **Name** choice list as **None**.

6. From the form context menu, select **Save**.
7. In the **Requires role** related list, click **Edit**.
8. Use the slushbucket to add the ODBC role, and then click **Save**.
9. Click **Submit**.

Installing the ODBC driver

Review setup requirements, download the ODBC driver installer, and install the ODBC driver to a computer.

You can install the ServiceNow ODBC driver on Microsoft Windows computers. To install the ODBC driver, set up an ODBC user in your ServiceNow instance, then download and install the ODBC driver. If you already have the ODBC driver installed, you can upgrade to the newest version.

- [ODBC driver installation requirements and supported software](#)

Install the ServiceNow ODBC driver on Microsoft Windows computers.

- [Download and install the ODBC driver](#)

Download the ODBC driver from the ServiceNow Knowledge Base and install the driver for the first time.

- [Upgrade the ODBC driver](#)

If you have previously installed an older version of the ODBC driver, run the installer to uninstall the previous version, and then run the installer again to upgrade.

- [Install an ODBC driver patch](#)

Use ServiceNow patches to install incremental ODBC fixes that occur between major ODBC releases.

Install the ServiceNow ODBC driver on Microsoft Windows computers.

Installation requirements

Before installing the ODBC driver, ensure that your configuration meets these requirements.

Installation requirements

Category	Requirement
Active user	<p>The user record on the instance used to perform the queries.</p> <p>Note: The account used to connect to the instance via the ODBC driver must be defined on the instance. Accounts using single sign-on are not supported by the ODBC driver.</p>
The soap_query role	<p>The user you use to query the database must have the soap_query role if the instance uses the glide.soap.strict_security high security setting.</p>

Category	Requirement
	<p>Warning: Do not enable WS-Security for all SOAP requests by setting the <code>glide.soap.require_ws_security</code> system property. It is incompatible with the ODBC driver. Enabling this setting blocks both ODBC driver and MID Server connections. Instead, use basic authentication.</p>
Target Table ACLs	The user you use to query the database must have read access for the tables that you want to query. For example, a user with the <code>itil</code> role can read task tables, such as <code>Incident</code> .
Target Table Web Service Access	The table you want to query must allow web service interaction. You can enable web service interaction using the application access settings.
Java Runtime Environment (JRE)	JRE must be enabled on your instance.
Operating System	<p>The ServiceNow ODBC driver supports installation on the following operating systems:</p> <ul style="list-style-type: none">• Windows XP• Windows Server 2003• Windows Server 2008• Windows Server 2012• Windows Server 2016

Category	Requirement
	<ul style="list-style-type: none">Windows VistaWindows 7Windows 8Windows 10
Hardware	<ul style="list-style-type: none">RAM: 1 GB minimumDisk space: 135 MB for installation, 200 MB for writing cache files during usage
Account	The Windows account used for the installation must have local Administrator rights to install an ODBC driver.
Networking	During usage, the ODBC driver requires HTTPS (port 443) connectivity to the ServiceNow instance. The communication between the ODBC driver and the ServiceNow instance uses standard SOAP web services.
End User License Agreement	Read the End User License Agreement for the ServiceNow ODBC driver.

For more information, see [Application access settings](#).

Supported software

The following table lists the operating systems and reporting applications compatible with each version of the ODBC driver.

Supported software

Driver Version	Operating System	Microsoft Excel	Microsoft SQL Server	Crystal Reports	Tableau	Informatica
1.0.13 and later	<ul style="list-style-type: none"> • Windows XP SP2 • Windows Vista • Windows 7 • Windows 8.x • Windows 10 • Windows Server 2003 • Windows Server 2008 • Windows 	<ul style="list-style-type: none"> • 2007 • 2010 • 2013 • 2016 	<ul style="list-style-type: none"> • 2008 • 2012 • 2014 	<ul style="list-style-type: none"> • 2008 • 2011 • 2013 	<ul style="list-style-type: none"> • 8.1 • 8.2 • 8.3 • 9.0 	<p>The ODBC driver provides only basic support for Informatica. Use the ODBC driver with Informatica only for simple operations. Thoroughly test integrations with Informatica before using them in a production environment.</p>

Driver Version	Operating System	Microsoft Excel	Microsoft SQL Server	Crystal Reports	Tableau	Informatica
	Server 2008 R2 <ul style="list-style-type: none"> • Windows Server 2012 • Windows Server 2012 R2 • Windows Server 2016 					
1.0.9 - 1.0.12	<ul style="list-style-type: none"> • Windows XP SP2 • Windows Vista • Windows 7 					

Driver Version	Operating System	Microsoft Excel	Microsoft SQL Server	Crystal Reports	Tableau	Informatica
	<ul style="list-style-type: none"> • Windows 8.x • Windows Server 2003 • Windows Server 2008 • Windows Server 2008 R2 • Windows Server 2012 • Windows Server 2012 R2 					
1.0.8 and earlier	<ul style="list-style-type: none"> • Windows 				8.1	

Driver Version	Operating System	Microsoft Excel	Microsoft SQL Server	Crystal Reports	Tableau	Informatica
	• ws XP SP2 • Windo ws Vista • Windo ws 7 • Windo ws 8.x • Windo ws Server 2003 • Windo ws Server 2008 • Windo ws Server 2008 R2					

Download the ODBC driver from the ServiceNow Knowledge Base and install the driver for the first time.

Before you begin

Role required: admin

You must have administrator-level access for the Windows computer onto which you want to install the ODBC driver.

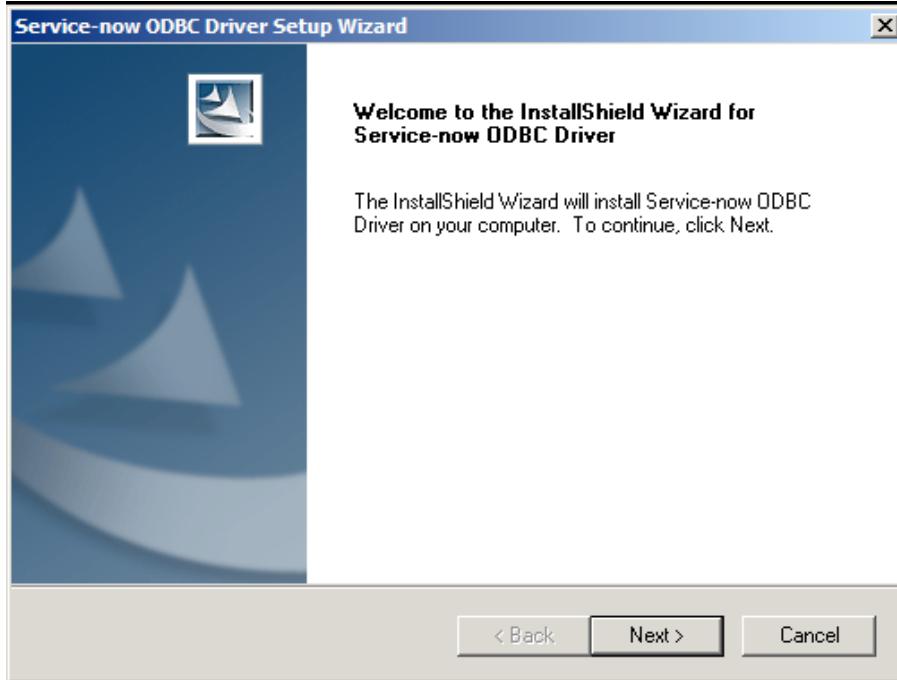
About this task

If this is the first time the driver is installed, the installer is in first time installation mode and prompts for the driver to be installed. Install only one version of the ODBC driver on a computer. If the ODBC driver was previously installed, the installer is in upgrade mode and prompts for removal of the previous driver first.

Note: Versions of the ODBC driver older than 1.0.7.3 are no longer supported.

Procedure

1. To download the latest ODBC drivers, click the following links:
 - 1.0.15 (32 bit) [download](#)
 - 1.0.15 (64 bit) [download](#)
2. Download the ODBC driver version compatible with your computer's operating system and the application you are using to query the database.
The ODBC driver is available as 32-bit or 64-bit. Most applications require the 32-bit ODBC driver even if the operating system is 64-bit.
3. Right-click the executable and select **Run as Administrator** to launch the installer.
The ODBC Driver Setup Wizard appears.



4. Click **Next**.
5. Read and accept the End User License Agreement.
6. The installer prompts for the **Installation Directory**. Select the target directory for installing the ServiceNow ODBC driver.
The default directory is C:\Program Files\Service-now\ODBC.
7. The installer prompts for the **Service Name**. Enter a name to identify the service (for example, ServiceNow_ODBC) and press **Next**.
8. The installer prompts for the **Java Virtual Machine Location**.
 - Browse to and select the directory where the jvm.dll file is located (usually C:\Program Files\Java\jre<version>\bin\server).
 - If you do not want to enter the JRE location at the time of the installation, click **Next**.
On the popup window, click **Yes** to enter the JRE location now, or click **No** to enter the JRE location later.

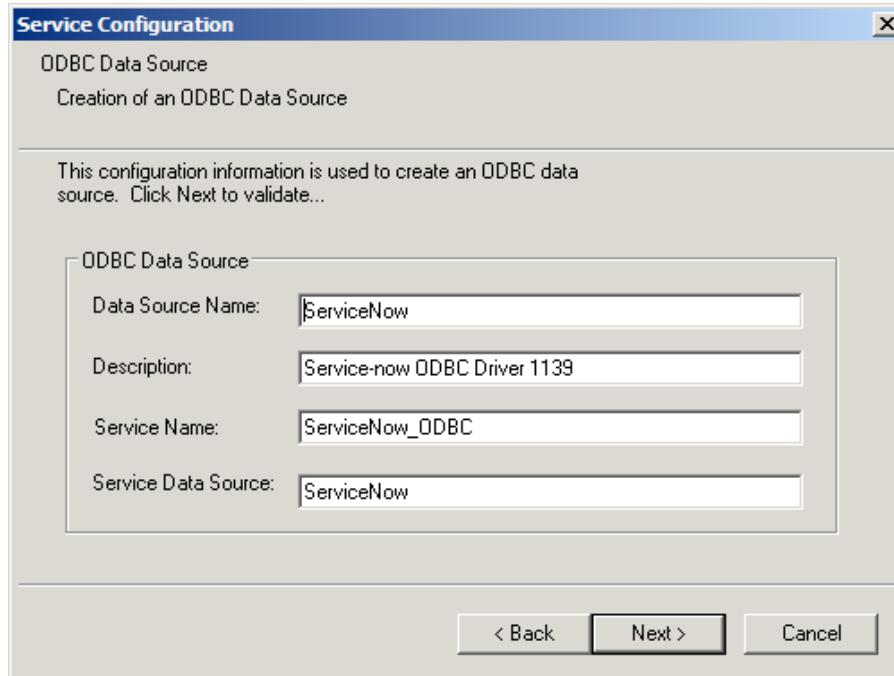
Important: You must specify the JRE location before the ODBC driver will work. After the installation, you can access the Management console as an administrator and navigate to **Services > <Service_Name> > Service Settings > IP Parameters** and enter the JRE location in the ServiceJVMLocation property.

Tip: For the configuration of a Java service, ServiceNow strongly recommends using the server jvm.dll file from the Java Development Kit (JDK).

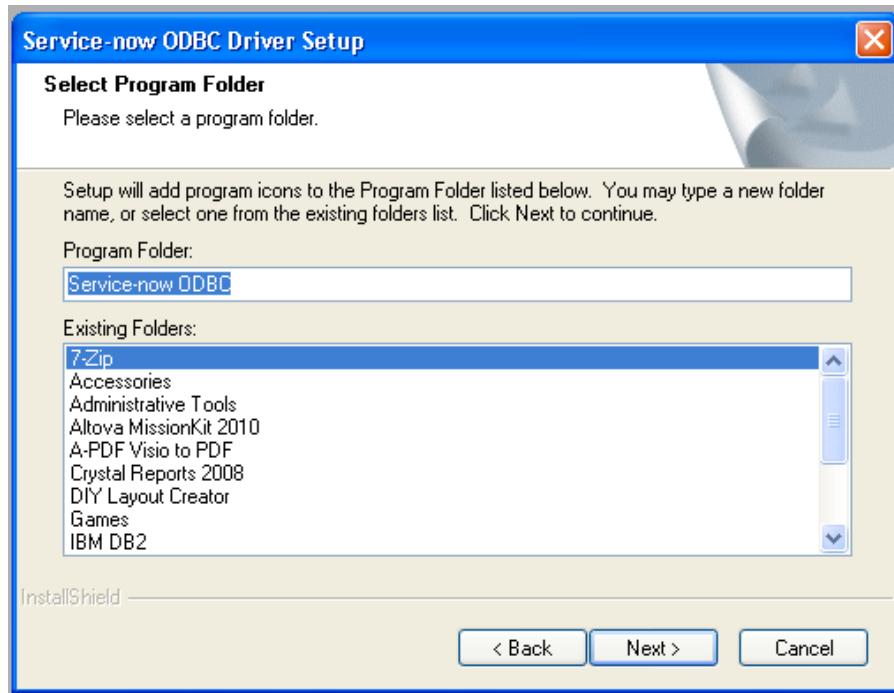
9. The installer prompts for the **ODBC Data Source** details. Specify the following parameters, which are required to create an ODBC data source that can be used to create a DSN.

- **Data Source Name:** A short name to identify this data source.
- **Description:** A short description of the driver. The driver's version number is appended to the end of this value.
- **Service Name:** Enter the same **Service Name** entered at step 7.
- **Service Data Source:** The name that can be selected in the **Service Data Source** field of the ODBC Administrator.

Usually the default values are appropriate.

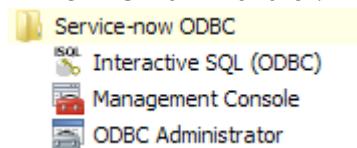


10. Select the **Program Folder** to create links for the driver. This is the program folder that appears under the **Start** menu.

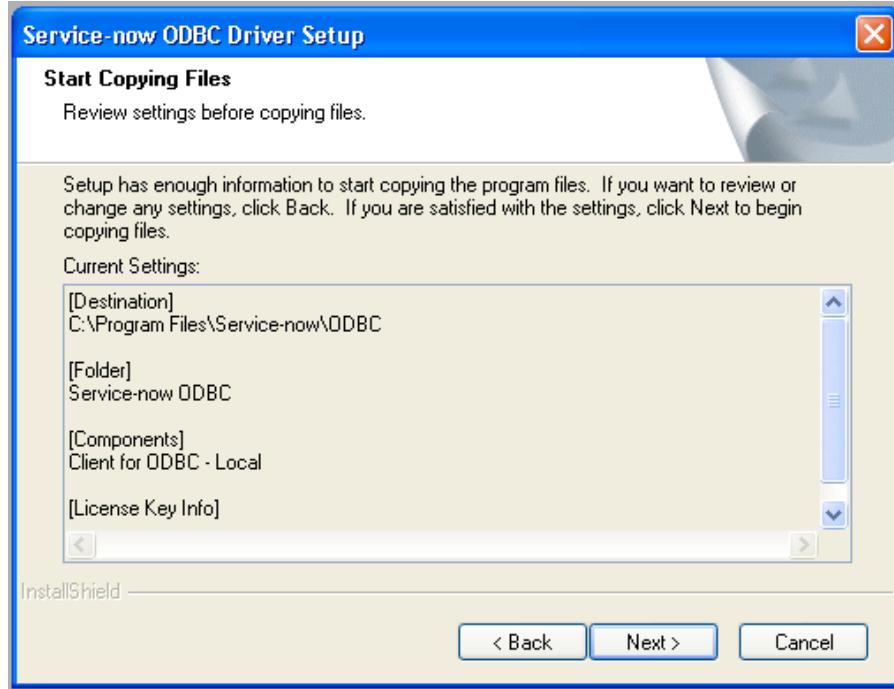


The installation creates the following links in the menu.

- **Interactive SQL (ODBC)**: An interactive SQL command window for directly testing SQL statements.
- **Management Console**: A Microsoft MMC snap-in for configuring default properties for the ODBC driver.
- **ODBC Administrator**: A Microsoft ODBC Administrator program.



The driver code is copied to the target folder.



A progress bar appears.

11. When prompted, click **Finish** to complete the installation.

If you have previously installed an older version of the ODBC driver, run the installer to uninstall the previous version, and then run the installer again to upgrade.

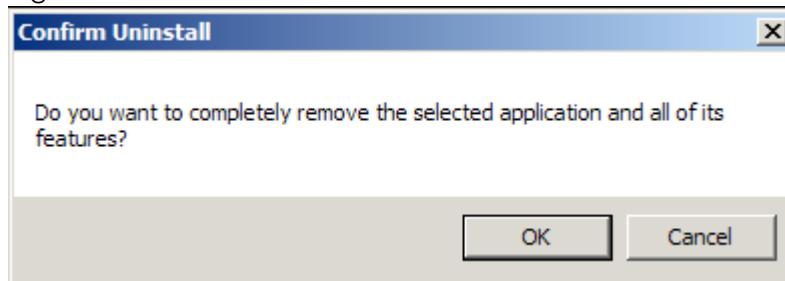
About this task

To check the build date and time of the ODBC driver, use **CheckVersion** located in the Service-Now\ODBC\ip\tools folder. This is an executable Windows host script that reports the build date and time of the current ODBC driver. Use it to assist ServiceNow Technical Support to determine which build of the ODBC driver is running. If the **CheckVersion** tool is absent, the ODBC driver is out of date; upgrade to the current version. To check the version of an older ODBC driver, see the previous version information.

Note: The ODBC installation also has a Service-Now\ODBC\tools folder, which is not the correct path for the CheckVersion tool.

Procedure

1. Right-click the executable and select **Run as Administrator**.

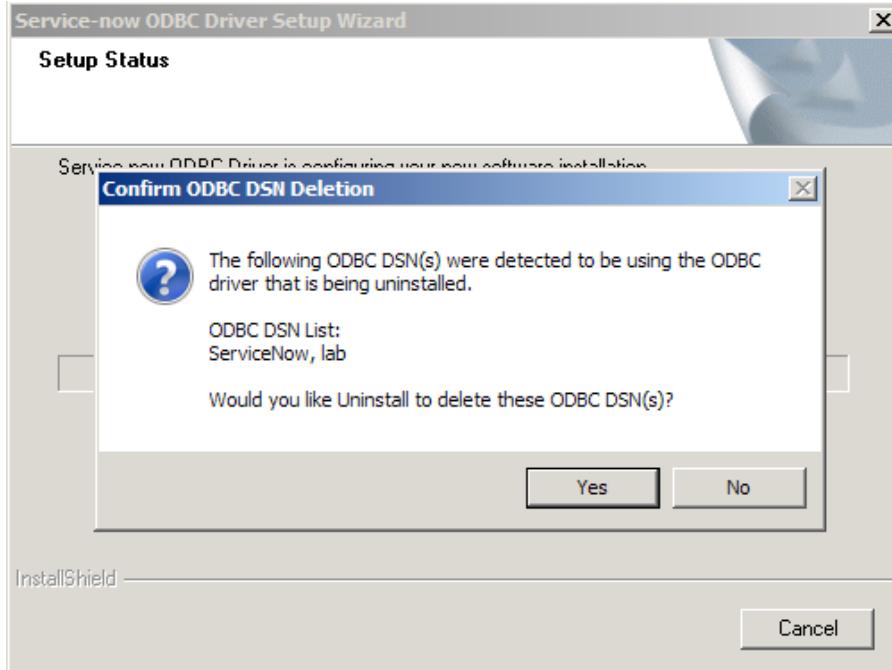


2. Click **OK** when prompted to uninstall the current driver, which is required for the upgrade.

A list appears, displaying the existing ODBC DSN names that you have previously created. You have the option to delete them.

3. Select **Yes** to remove all previous DSNs or **No** to keep them for use with the upgraded driver.

An ODBC DSN is a connection handle to use the ODBC driver in an application. For more information from Microsoft, see [Administer ODBC data sources](#).



4. After removing the previous ODBC driver, double-click the executable again to run the installer. Then, follow the steps in [Download and install the ODBC driver](#).

If you encounter errors when uninstalling the ODBC driver, see the [Troubleshooting uninstalling ODBC driver](#) knowledge article.

What to do next

After installing the ODBC driver, configure it to connect to your ServiceNow instance.

Use ServiceNow patches to install incremental ODBC fixes that occur between major ODBC releases.

Before you begin

Incremental ODBC patches contain only fixes to the ODBC driver and do not include fixes for third-party products such as Progress SDK. Third-party product fixes are included only in major ODBC release updates.

You must have administrator-level access for the Windows computer onto which you want to install the ODBC driver patch.

Before applying a new ODBC patch:

- Ensure that you have the Java Runtime Environment (JRE) already installed in your instance.
- Close any applications currently using the ODBC driver.

Procedure

1. Download and unzip the ODBC patch zip file from [KB0540707](#) in the ServiceNow Knowledge Base.
2. Extract the zip files into a folder named odbc-patch.
3. From the **Start** menu, right-click **Command Prompt** and select **Run as administrator**.
4. In the Command Prompt window, cd to the odbc-patch folder created in step 2, type update.bat, and press **Enter**.
5. The installer prompts for the installation directory of the ODBC driver. The default installation location is C:\Program Files\ServiceNow\ODBC.
 - If the ODBC driver is installed in C:\Program Files\ServiceNow\ODBC, enter **y** and press **Enter**.
 - If the ODBC driver is not installed in C:\Program Files\ServiceNow\ODBC, enter the complete path to the ODBC installation directory and press **Enter**.

The installer validates the installation location. If the location specified is not valid, the installer prompts for the correct location of the installed ODBC driver.

Once you enter a valid installation location, the installer checks for the installed version and compares it with the version of the patch.

- If the currently-installed version is the latest version, the installer aborts the installation process and displays the following message:

The version you are trying to install is same as the existing version. Installation aborted.

- If the currently-installed version is the same as the patch version, the installer aborts the installation process and displays the following message:

The current installed version is the most recent version. Installation aborted.

- If the patch version is the latest version, the installer continues with the patch installation. Once the installation is complete, you receive the following confirmation message:

Patch is successfully installed.

Configuring the ODBC driver

After installing the ODBC driver, configure it to connect to your ServiceNow instance and to communicate through a proxy server, if applicable, and set properties to control ODBC behavior.

- [Configure the ODBC driver](#)

Configure the ODBC driver to connect to your ServiceNow instance.

- [Configure the global DSN default](#)

Configure the global default used by all newly created DSNs.

- [Create a new DSN](#)

Use the ODBC driver and the ServiceNow data source to create an unlimited number of DSNs configured to connect with different instance URLs.

- [Specify a connection string](#)

You can specify a connection string instead of defining a DSN.

- [Configure the logging level of the ODBC driver](#)

Change the logging level of the ODBC driver.

- [Configure the ODBC driver for large data sets](#)

You can set two ODBC driver properties to deal with errors you receive when using queries that return large amounts of data.

- [Configure ODBC to use proxy servers](#)

The ODBC driver can be configured to route its HTTP SOAP requests via an HTTP proxy server.

- [Setting ODBC properties](#)

The following properties customize connectivity and optimize the query behavior of the ODBC driver.

Configure the ODBC driver to connect to your ServiceNow instance.

Before you begin

You must have administrator-level access for the Windows computer on which you want to configure the ODBC driver.

About this task

After the driver is installed, configure it for your instance. The driver is preconfigured to connect to <https://demoodbc.service-now.com> using the DSN ServiceNow. There are two ways to configure connectivity for the driver.

- Configure the global default used by all newly created DSNs.
- Configure each new DSN with its own connection.

Procedure

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC Management Console**.
2. Expand the Console Root tree to:
ServiceNow ODBC Manager\Manager\<installation location>\Services\ServiceNow_ODBC\Data Source Settings\ServiceNow\IP Parameters.
3. Double-click the **DataSourceIPProperties** attribute.

4. Change the **Value** to the URL of your instance, such as `https://<instance>.service-now.com`.
If integrating the ODBC driver with Edge Encryption, change the **Value** to the URL of your encryption proxy. See [Edge Encryption ODBC driver integration](#) for more information.
5. Click **OK**.

Configure the global default used by all newly created DSNs.

Before you begin

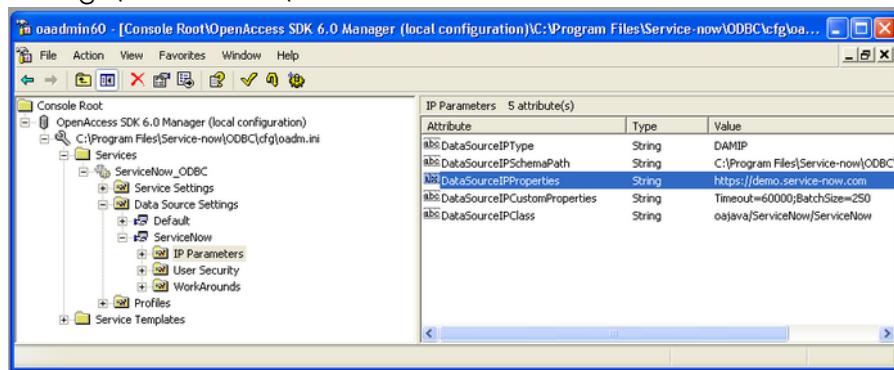
You must have administrator-level access for the Windows computer on which you want to configure the global DSN default.

About this task

A default DSN is preloaded with the ODBC driver installation ServiceNow data source. This preloaded DSN connects using the default connection URL, which is set to `https://demo.service-now.com`. To change the global default for the instance URL, do the following.

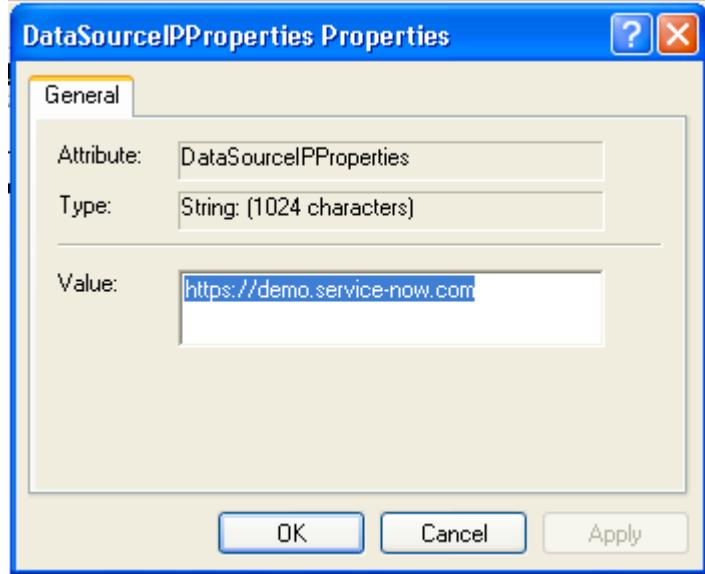
Procedure

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC > Management Console**.
2. Expand the Console Root tree using the following path:
`OpenAccess SDK 6.0 Manager\<installation location>\Services\ServiceNow_ODBC\Data Source Settings\ServiceNow\IP Parameters`



3. Double-click the **DataSourceIPProperties** attribute for the ServiceNow data source setting to open the Properties dialog box.
4. Change the value to the URL of your instance, using the following format, and then click **OK**:

`https://<your instance>.service-now.com`



Use the ODBC driver and the ServiceNow data source to create an unlimited number of DSNs configured to connect with different instance URLs.

Before you begin

You must have administrator-level access for the Windows computer on which you want to create a new DSN.

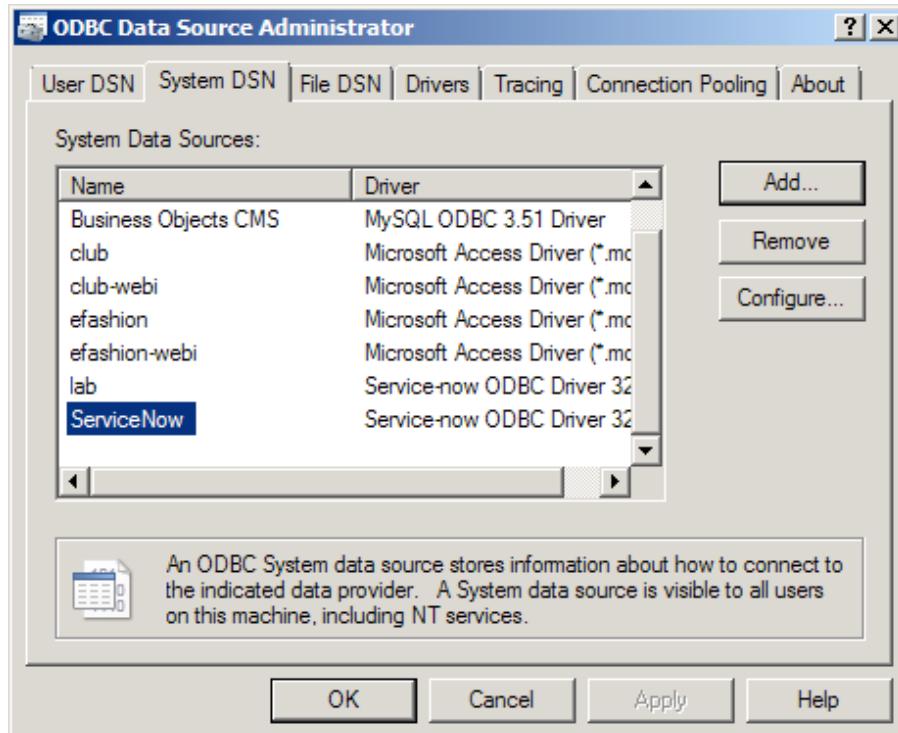
About this task

Select the target instance for your ODBC connection by DSN name. As an option during installation or upgrade, you can elect to keep the DSNs when you uninstall.

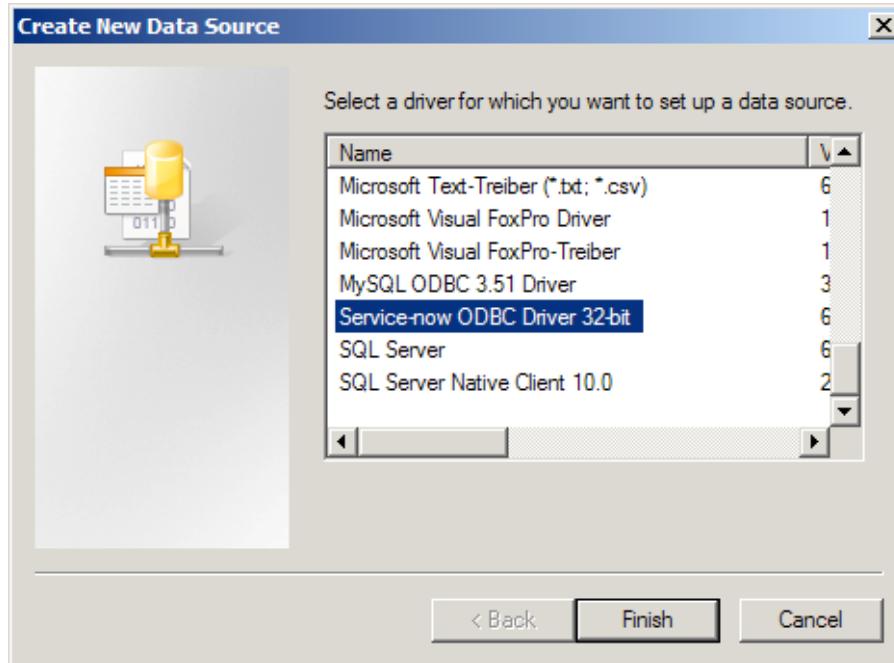
Instead of creating a new DSN, you can [Specify a connection string](#) to connect with different instance URLs.

Procedure

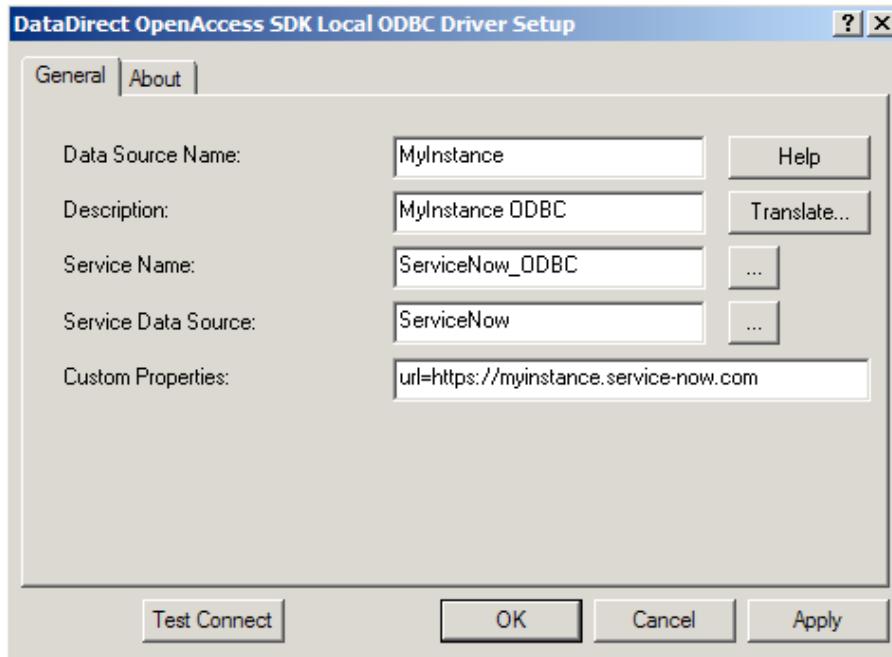
1. In Windows, navigate to **Start > Programs > Service-now ODBC > ODBC Administrator.**



2. To create a system DSN, select the **System DSN** tab, and then click **Add**.



3. Select ServiceNow **ODBC driver 32-bit** from the list, and then click **Finish**.
4. Configure the driver and its connection URL by specifying the url= parameter value in the **Custom Properties** field. For example:
url=https://myinstance.service-now.com



5. Click **OK**.

Result

You can now use the new driver.

You can specify a connection string instead of defining a DSN.

Before you begin

You must have administrator-level access for the Windows computer on which you want to specify a connection string.

About this task

This is an alternative method of connecting with different instance URLs.

See also [Create a new DSN](#).

A connection string must follow this format:

```
Driver=ServiceNow ODBC driver 32-bit;ServiceName=ServiceNow_ODBC;UID=youruser;PWD=yourpassword;ServerDataSource=ServiceNow
```

```
iceNow;CustomProperties=url=https://<instance>.service-now.com
```

The driver name varies depending on whether you use the 32-bit or 64-bit version of the ODBC driver. To determine your driver name, do the following.

Procedure

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC > ODBC Administrator**.
2. Select the **System DSN** tab.
3. Note the value in the **Driver** column for the ServiceNow data source.

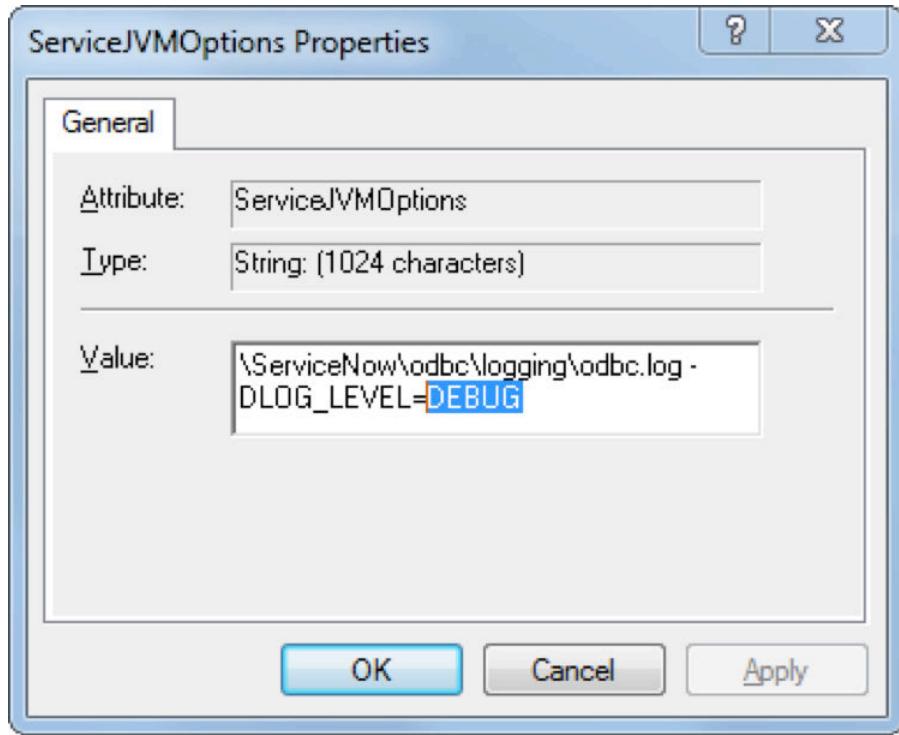
Change the logging level of the ODBC driver.

Before you begin

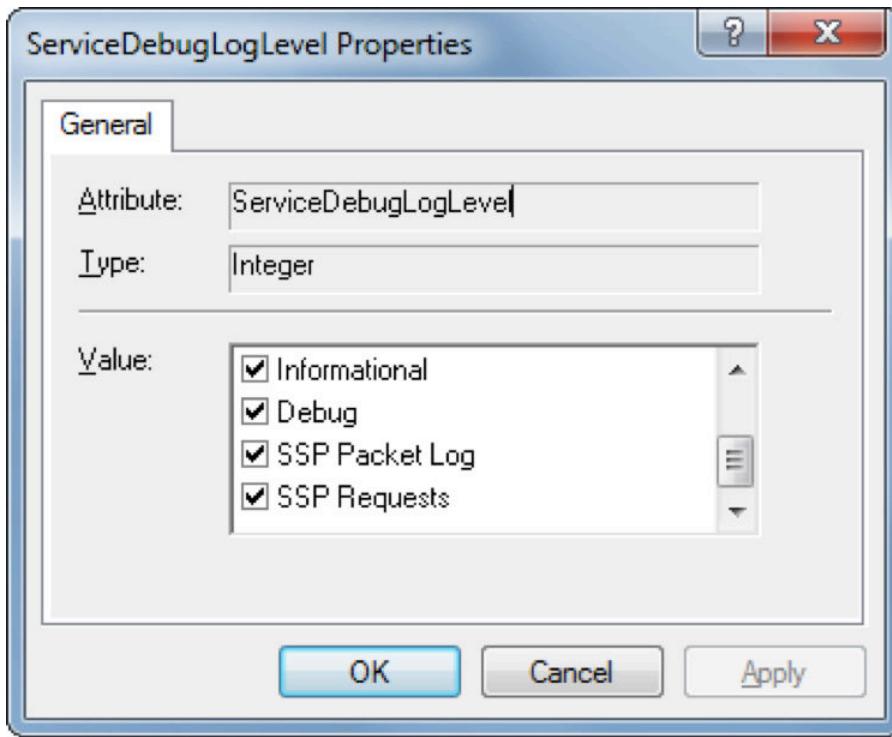
You must have administrator-level access for the Windows computer on which you want to configure the logging level.

Procedure

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC > Management Console**.
2. [ODBC version 1.0.8] Within the management console, navigate to **<your_installation_directory> > Services > ServiceNow_ODBC > Service Settings > IP Parameters**.
3. [ODBC version 1.0.8] Change the value of the **ServiceJVMOptions** attribute to the desired logging level.



4. Within the management console, navigate to
<your_installation_directory> > Services > ServiceNow_ODBC > Service Settings > Logging.
5. Change the value of the **ServiceDebugLogLevel** by selecting all available check boxes.



6. In Windows, navigate to **Start > Programs > ServiceNow ODBC > ODBC Administrator**.
7. In the ODBC Administrator, select the **Tracing** tab.
8. Navigate to the path in the **Log File Path** field and delete the old log file, if it exists.
9. Click **Start Tracing Now**.
10. Enable SOAP debugging for your ServiceNow instance.

Related tasks

- [Enable debug logging](#)

Related concepts

- [Debug incoming SOAP envelope](#)

You can set two ODBC driver properties to deal with errors you receive when using queries that return large amounts of data.

Before you begin

You must have administrator-level access for the Windows computer on which you want to configure the ODBC driver for large data sets.

About this task

These properties are set using the ODBC Management Console available on Windows operating systems. For more information, see [ODBC management console properties](#).

Procedure

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC > Management Console**.
2. Navigate to **Data Source Settings > ServiceNow > IP Parameters**.
3. Set the **Timeout** property to be more than the `glide.soap.request_processing_timeout.odbc` value.
4. Navigate to **Services > Service Settings > SQL Engine Parameters**.
5. Increase the **ServiceSQLDiskCacheMaxSize** property.
Typically, when running a query that returns 50,000 rows, the default value of 200 must be increased.

The ODBC driver can be configured to route its HTTP SOAP requests via an HTTP proxy server.

Before you begin

You must have administrator-level access for the Windows computer on which you want to configure ODBC to use proxy servers.

About this task

Setting up a proxy server gives you the option to control access to the ServiceNow instance from the proxy server, and potentially allows a network configuration that can monitor usage statistics. However, because the proxy server intercepts the ODBC driver's requests to your ServiceNow instance, it will degrade the performance of the driver.

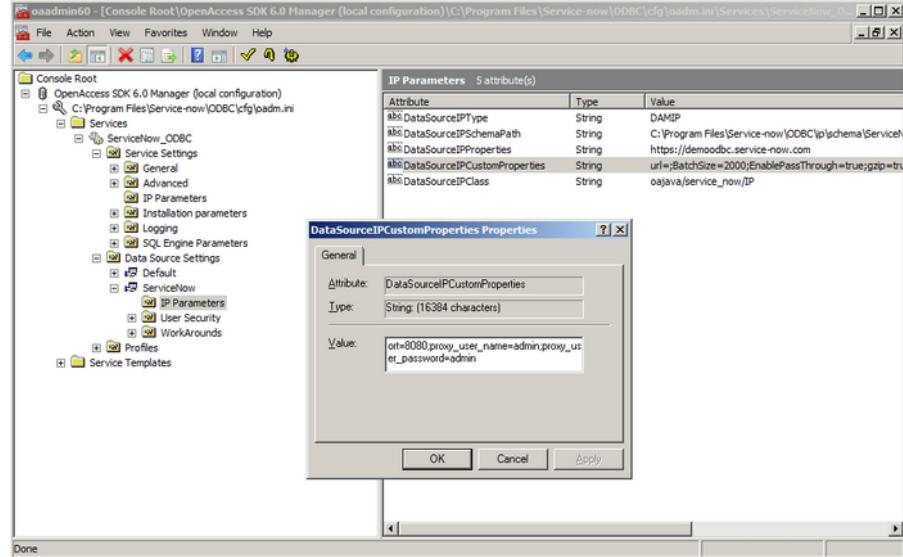
Note: This feature is recommended for use with ODBC driver builds dated 7/15/2011 or later.

To enable the use of proxy servers, the custom properties for proxy server settings must be defined first for the data source. After that, these properties can be overridden by specific ODBC DSNs.

Procedure

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC > Management Console**.
2. Expand the Console Root tree to:
ServiceNow ODBC Manager\Manager\<installation location>\Services\ServiceNow_ODBC\Data Source Settings\ServiceNow\IP Parameters.
3. Double-click the **DataSourcelpCustomProperties** attribute.

ODBC proxy



- Set the following custom properties to configure the ODBC proxy server.

Configure ODBC to use proxy servers

Property name	Description	Example
proxy_host	The proxy server host name or IP address.	proxy.company.com
proxy_port	The proxy server port number.	8080
proxy_user_name	The proxy server user name or id, used in an authenticating proxy configuration.	odbc_user
proxy_user_password	The proxy server user password, used with the proxy_name value in an authenticating proxy configuration.	*****

The following properties customize connectivity and optimize the query behavior of the ODBC driver.

ODBC administrator properties

These properties are specified in the ODBC Data Source Administrator for the DSN or in the **Custom Properties** field of the login dialog box.

ODBC administrator properties

Property Name	Description	Default
BatchSize	During fetching of results from the instance, this batch size configures the number of records to fetch for every request. Typically, the default is an optimal number for normal sized rows. If an error occurs during fetching of records that indicates this value should be lowered, you can modify it to optimize memory usage versus performance.	2000
url	This is the ServiceNow instance URL or endpoint. It should indicate the URL to the ServiceNow instance you want to connect to.	https://demo.service-now.com
EnablePassThrough	During processing of aggregate functions, enabling pass through mode allows directly	true

Property Name	Description	Default
	calling Aggregate Web Service for optimized and speedy response. Whenever possible, this mode should be left enabled.	
debug	By default, debugging messages are not produced. Set debug to true when you operate the ODBC driver from the ISQL console window to write all HTTP-related network communication traffic to the console window. When using this option, set gzip to false so that data is not compressed. Otherwise, the data is unreadable.	false
gzip	By default, data sent over the network is compressed. Set gzip to false when using the debug parameter to write network communication to the ISQL console so that data is not compressed.	true
timeout	Specifies the socket inactivity timeout value in seconds.	175

Property Name	Description	Default
retries	Number of times to retry the failing request in the event of a socket timeout error.	0
mode	The query mode used to parse complex where clauses. You can configure the ODBC driver query mode to use either AND or OR operators. While the OR operator provides the greatest compatibility with complex queries, the AND operator is usually more efficient and results in fewer database operations.	or
EnableDBSchema	The ODBC driver issues a database schema request to retrieve table names from the instance. This functionality is enabled by default so reporting applications such as Microsoft Excel can display a list of tables to query from. Disabling this property may improve the performance of the first query sent from a reporting application, especially if the instance has	true

Property Name	Description	Default
	a large number of tables.	
ExtendedSchemaCache	The ODBC driver caches the database schema for each connection. When a new connection is created, the driver clears the database cache and queries the database schema from the instance again. This behavior is beneficial when connecting to different data source, or when modifying the table schema. When querying a single data source with a consistent schema, enable this property to avoid sending unnecessary schema requests, including when EnableDBSchema is true .	false
LegacyDurationTimeZone	The ODBC driver returns timer and duration field values in the UTC timezone by default, starting with the 1.0.10 version. When this property is true , the ODBC driver returns timer and duration field values using the display value, as shown in the	false

Property Name	Description	Default
	UI. This property can be used to preserve compatibility with legacy integrations that depend on the display value. See KB0583982 for details about this behavior.	

If you need to use more than one of these properties in your connection, concatenate the settings with a semicolon (;) delimiter. For example, the following string sets the URL to a specific instance and changes the batch size to 200 records.

```
url=https://demo1234.service-now.com;BatchSize=200
```

ODBC management console properties

You can access these properties from the ODBC Management Console available in the Windows Start menu at **ServiceNow ODBC > Management Console**.

ODBC management console properties

Property name	Description	Default
ServiceJVMOptions (Services\Service Settings\IP Parameters)	JVM command line properties and option. For example, to change the maximum Java heap size, modify the -Xmx150m parameter.	-Xms64m -Xmx150m
DataSourceProperties (OpenAccess SDK 6.0 Manager\<installation location>\Services\ServiceNow_ODBC\DataSource)	Global default of the instance URL for all ODBC connections. For more flexibility, you may also create new DSNs with default URL configurations.	https://demo.service-now.com

Property name	Description	Default
Settings\ServiceNow\Parameters		
ServiceSQLDiskCacheMaxSize (Services\Service Settings\SQL Engine Parameters)	Specifies the maximum size of the disk cache files. Increase this value when you see Disk Cache file size limit has reached errors.	200
Timeout (Data Source Settings\ServiceNow\Parameters)	Specifies the socket inactivity timeout value in seconds. Increase this to a value greater than glide.soap.request_processing_timeout.odbc when you see GetKeys failed (Socket timeout) errors.	175
ServiceJVMLocation	Contains the JRE location used by the ODBC driver.	There is no default value for this property. You are prompted to enter the JRE location at the time of installation.

Service JVM options

You can specify these values within the ServiceJVMOptions parameter in addition to standard JVM arguments such as -Xmx.

Service JVM options

Option	Description	Default
-DLOG_FILE_NAME	The location of the ODBC log file. This property is available starting with the ODBC driver 1.0.7.1 release.	\${user.home} \AppData\Local\ServiceNow\odbc\logging\odbc.log
-DLOG_LEVEL	The logging level used when writing to the ODBC log file. You can specify the logging level using Logback levels, such as TRACE, INFO, or ERROR. This property is available starting with the ODBC driver 1.0.8 release.	INFO

Instance properties

An administrator can configure these properties by adding a property or modifying an existing one in the ServiceNow instance.

Instance properties

Property name	Description	Default
glide.db.max.aggregates	The maximum number of rows returned by aggregate functions.	100000
glide.db.max_view_records	The maximum number of rows returned by a database view.	10000

Test the ODBC driver

After configuring the ODBC driver, test that the driver can connect to the base instance as the ODBC user and can query data from a target table.

Before you begin

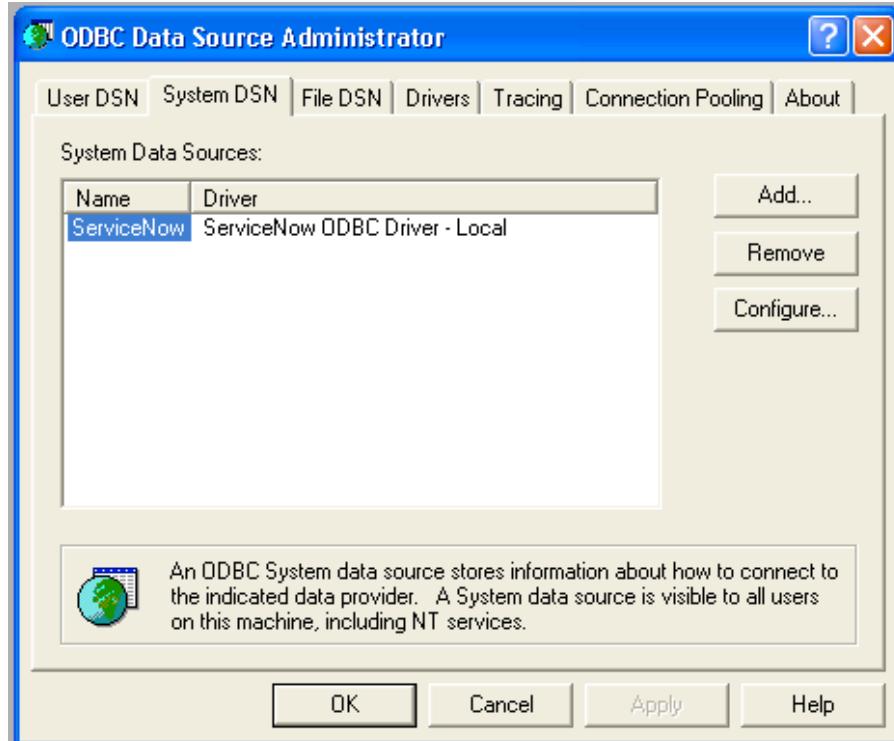
You must have administrator-level access for the Windows computer on which you want to test the ODBC driver.

About this task

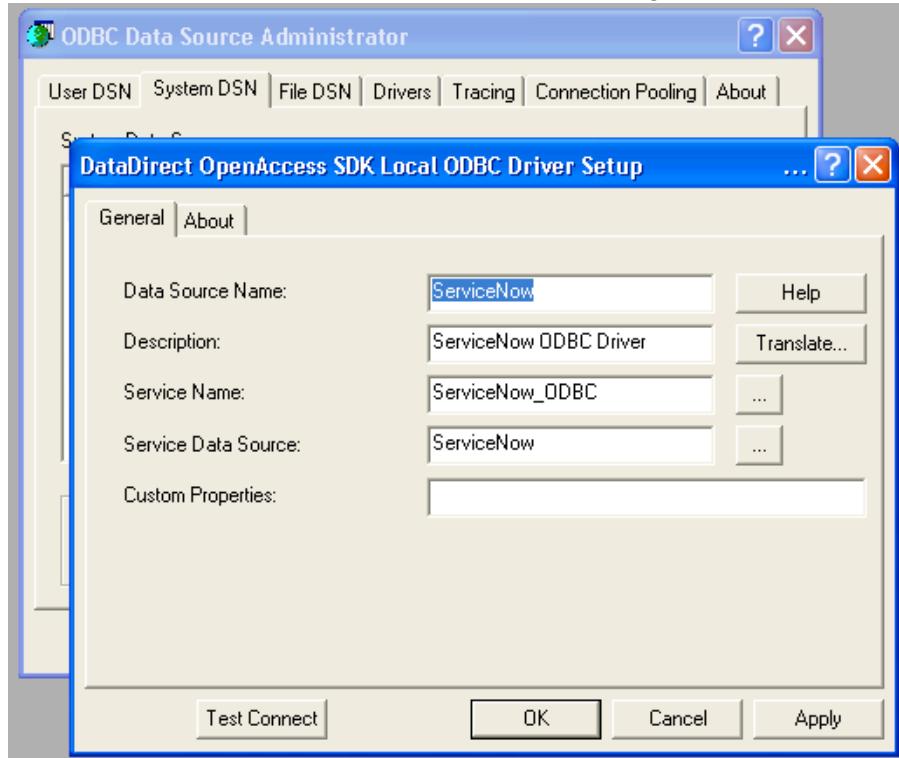
To test the connection, run the ODBC Administrator program.

Procedure

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC > ODBC Administrator**.
The ServiceNow ODBC data source is installed as a system data source.
2. Select the **System DSN** tab, and then select the **ServiceNow** data source.
3. Click **Configure**.

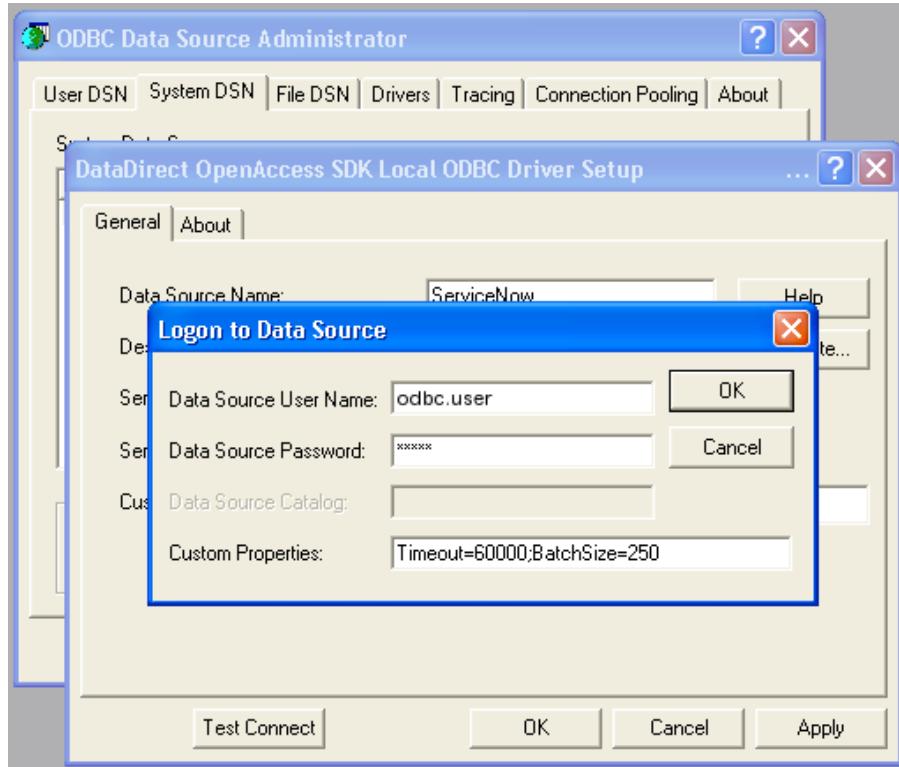


4. Click **Test Connect** in the ODBC driver Setup dialog box.



5. Enter the login credentials.

These are the usual ServiceNow base system login credentials for the ODBC user you created.



6. Click **OK** to log in to the data source.

7. Click **OK** again when the success message appears.

- [Enable debug logging](#)

If you experience unexpected behavior when using the ODBC driver, you can enable debug logging and generate debug logs to help identify the issue.

- [Test a query](#)

To verify that the user has the appropriate permissions to send requests to the instance using ODBC, run a query using Interactive SQL.

- [ODBC troubleshooting](#)

Review these troubleshooting resources to resolve issues with the ODBC driver.

Related tasks

- [Configure the ODBC driver](#)

If you experience unexpected behavior when using the ODBC driver, you can enable debug logging and generate debug logs to help identify the issue.

About this task

Debug logs can be useful when submitting an incident with Customer Service and Support.

When you enable debug logging, note the version and bitness (32 bit or 64 bit) of the installed ODBC driver, the Windows operating system, and the client application you are using with the ODBC driver.

To generate debug logs, follow these steps.

Procedure

1. Close all active applications that may use the ODBC driver.
2. Navigate to one of these paths, based on your operating system.
 - For Windows 7:
C:\Users\<user_name>\AppData\Local\ServiceNow\odbc\logging
 - For Windows XP and earlier:
C:\Program Files\ServiceNow\ODBC\%LOCALAPPDATA%\ServiceNow\odbc\logging
3. Delete any existing log data to ensure that you log only relevant information.
4. Run a query that produces the unexpected behavior, then immediately close the application and review the log files.

Related tasks

- [Configure the logging level of the ODBC driver](#)

To verify that the user has the appropriate permissions to send requests to the instance using ODBC, run a query using Interactive SQL.

About this task

For testing, use a query that returns exactly one record, such as a query using the **Number** value of a record.

Procedure

1. In the base system instance, navigate to **Incident > All**.
2. Record the **Number** of an incident record.
3. On the computer where the ODBC driver is installed, navigate to **Start > Programs > ServiceNow ODBC > Interactive SQL**.
4. Enter connect "odbc.user"**password"@ServiceNow and press **Enter**.
5. Enter the following text, substituting the incident number you recorded.
`select short_description from incident where
number='<incident number>';`
6. Press **Enter**.

Result

The instance should respond with the short description of the incident record.

Review these troubleshooting resources to resolve issues with the ODBC driver.

For troubleshooting information, see the Knowledge Base articles [troubleshooting ODBC driver issues](#) and [troubleshooting common ODBC error messages](#).

Related tasks

- [Create an ODBC user account and assign the odbc role](#)
- [Define an ACL rule for the odbc role](#)
- [Configure the ODBC driver](#)
- [Test the ODBC driver](#)

Related concepts

- [Getting started with ODBC](#)

ODBC behavior

After testing the ODBC driver you can use it to query your instance database from a variety of client applications.

ODBC aggregate functions

The ODBC driver attempts to download data and apply aggregate functions locally. The ODBC driver supports the following aggregate functions.

- COUNT
- SUM
- MIN
- MAX
- AVG

Activate the Aggregate Web Service plugin to improve the performance of aggregate queries through the ODBC driver.

ODBC date and time values

The instance and the machine on which the ODBC driver is installed may use two different time zones. Date and time values returned by the ODBC driver are in the local time zone of the application using the driver, not the ServiceNow instance time zone.

Ensure that you query in accurate time zones for both the instance and the machine that hosts the ODBC driver. GlideRecord performs filtering based on the instance time zone, and the ODBC client is filtered based on the Windows time zone.

For example, an instance is in Central Standard Time (CST), and the ODBC driver is installed on a machine that is in Pacific Standard Time (PST). An incident is created on the instance at 2014-05-20 10:00:00, and the time that the incident was created is displayed in the UI as 10:00:00 for users in both time zones. However, in order to successfully query this incident by creation date and time, a user on the machine in PST must query 2014-05-20 08:00:00 instead of 2014-05-20 10:00:00.

Duration and timer type fields are returned using the UTC timezone, starting with ODBC version 1.0.10. See [KB0583982](#) for details about this change.

ODBC display values

Some examples of how to use and work with ODBC display values are shown below.

- Display values in **Choice** and **Reference** columns:

When querying a column of type **Choice**, **Reference**, **Duration**, or **Timer**, an additional column with the prefix **dv_** is available that contains the display value. For example, you can select **dv_caller_id** to return the **sys_user.name** display value of the reference field from an incident record without making another request to the sys_user table.

Return the display value

```
ISQL> select number, dv_caller_id, caller_id from incident;
number  dv_caller_id    caller_id
INC00000009      Rick Berzle      5137153cc611227c000bbd1bd8cd2006
INC00000010      Fred Luddy      5137153cc611227c000bbd1bd8cd2005
INC00000011      Don Goodliffe   9ee1b13dc6112271007f9d0efdb69cd0
INC00000012      Don Goodliffe   9ee1b13dc6112271007f9d0efdb69cd0
INC00000013      Joe Employee    681ccaf9c0a8016400b98a06818d57c7
INC00000014      Bow Ruggeri     f298d2d2c611227b0106c6be7f154bc8
INC00000015      Fred Luddy      5137153cc611227c000bbd1bd8cd2005
INC00000016      Bow Ruggeri     f298d2d2c611227b0106c6be7f154bc8
INC00000017      Joe Employee    681ccaf9c0a8016400b98a06818d57c7
INC00000018      Taylor Ureeland  46bac3d6a9fe1981005f299d979b8869
INC00000019      Fred Luddy      5137153cc611227c000bbd1bd8cd2005
```

- Display values in filter conditions:

Display values can also be used in a filter condition. The ODBC driver optimizes the query condition and processes the filter on the server, for example, querying on the display value of **sys_user** for the **caller_id** field of an incident by using the **dv_caller_id** field name.

Display values in filter conditions

```
ISQL> select number, dv_caller_id, caller_id from incident where dv_caller_id = 'Fred Luddy';
number    dv_caller_id   caller_id
INC00000010      Fred Luddy      5137153cc611227c000bbd1bd8cd2005
INC00000015      Fred Luddy      5137153cc611227c000bbd1bd8cd2005
INC00000019      Fred Luddy      5137153cc611227c000bbd1bd8cd2005
INC00000027      Fred Luddy      5137153cc611227c000bbd1bd8cd2005
```

- Display values in aggregate queries:

Aggregate queries can also take advantage of display values if you specify them in the group by or where clause, for example, grouping on the **caller_id** field of an incident, as well as specifying a filter for it. The query is optimized by passing through to the server.

Display values in aggregate queries

```
ISQL> select count(*), dv_caller_id from incident where dv_caller_id is not null group by dv_caller_id;
COUNT(*)          dv_caller_id
1                Beth Anglin
4                Bow Ruggieri
4                Bud Richman
2                Carol Coughlin
1                Charlie Whitherspoon
1                Christen Mitchell
1                David Loo
3                Don Goodliffe
4                Fred Luddy
1                Jerrod Bennett
1                Joe Employee
1                Luke Wilson
1                Margaret Grey
1                Natasha Ingram
5                Rick Berzle
1                Sam Sorokin
1                Taylor Vreeland
```

Querying table and column names

You can get a list of accessible tables and columns based on the read ACLs for the querying user.

- The following query will return the names of all tables for which the querying user has read access:

```
select * from oa_tables;
```

- After you know the name of the table you want to query, you can query the names of all columns for which the user has read access. The querying user must have read access for both the table and the columns.

```
select * from oa_columns where table_name='table_name';
```

Note: The oa_tables and oa_columns tables are internal ODBC tables. These tables are accessible only via the ODBC driver.

- [Increase the field length in SQL queries](#)

The ODBC driver limits the field length in SQL queries to the maximum length defined by the ServiceNow dictionary entry. You can increase the maximum field length to avoid truncating data.

The ODBC driver limits the field length in SQL queries to the maximum length defined by the ServiceNow dictionary entry. You can increase the maximum field length to avoid truncating data.

Before you begin

Role required: admin

About this task

If the data coming from the ODBC source exceeds the field size of the dictionary entry, ServiceNow truncates the query output to fit the field size. If your data is truncated, you can do the following.

Procedure

1. Increase the maximum length in the dictionary entry for the field in question.
2. Reconnect the ODBC driver to pick up the change.

Note: By default, the ODBC driver uses the VARCHAR data type to store query string output. When strings become very large (roughly 16000 characters), the ODBC driver uses the LONGVARCHAR data type instead. It is important to keep in mind, however, that the LONGVARCHAR data type has a more limited set of SQL commands that can be executed on it. For example, it does not support queries using scalar data.

Related tasks

- [Use Interactive SQL with ODBC](#)

ODBC and client applications

See the following pages for examples of how to use the ODBC driver to create data sources from other applications.

- [Use Interactive SQL with ODBC](#)

Run the Interactive SQL application for quick verification of connectivity and to test query results without using a full application.

- [ODBC driver in SQL Server](#)

Use the ServiceNow ODBC driver in SQL Server as a Linked Server.

- [Use the ODBC driver in Excel](#)

After installing the ODBC driver and its associated DSN, use it in Excel as a data source provider.

- [Use the ODBC driver in Crystal Reports](#)

After installing the ODBC driver and its associated DSN, use it in Crystal Reports as a data source provider.

Run the Interactive SQL application for quick verification of connectivity and to test query results without using a full application.

1. In Windows, navigate to **Start > Programs > ServiceNow ODBC > Interactive SQL (ODBC)**.
2. Enter the following command to connect to the base instance. Select the appropriate user credentials in the format: ID*password@DSNName. The password cannot contain special characters.

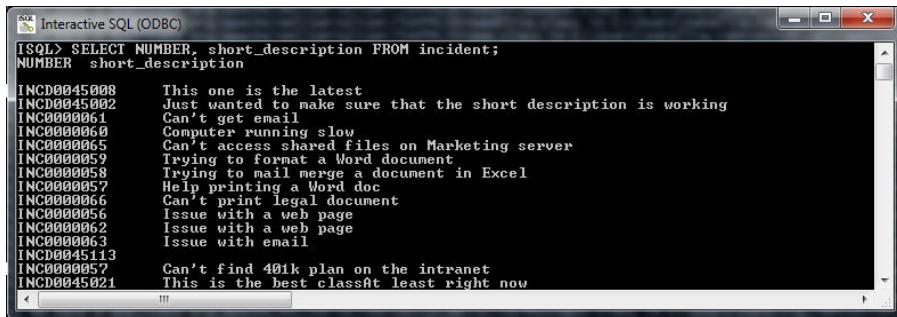
```
CONNECT odbcuser*password@ServiceNow
```



3. Issue a SELECT SQL command, such as:

```
SELECT NUMBER, short_description FROM incident;
```

Make sure to include the semicolon at the end of your query statement. You will be presented with a 'Cont>' prompt otherwise.



- Specify the maximum number of rows returned

By default, ServiceNow only returns 100 rows of data with each iSQL query. If you need to return more rows of data, set the maxrows parameter for the iSQL session.

- SQL support

The ODBC driver embeds a third party SQL/ODBC engine from DataDirect, a division of Progress Software.

By default, ServiceNow only returns 100 rows of data with each iSQL query. If you need to return more rows of data, set the maxrows parameter for the iSQL session.

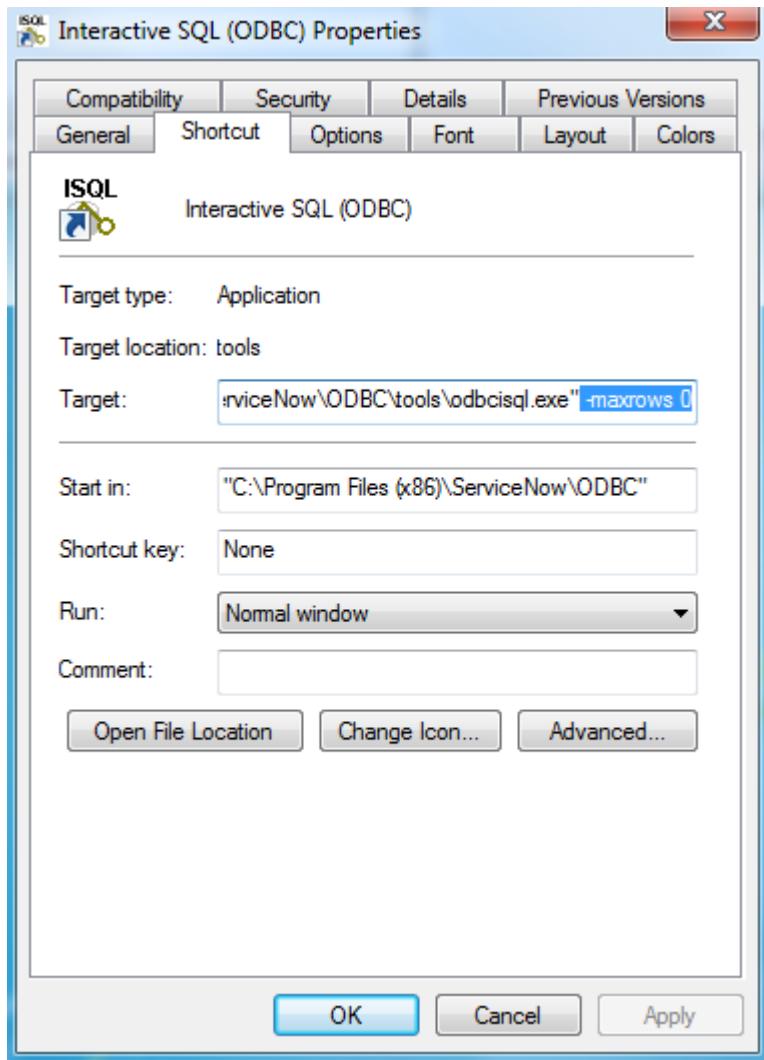
To return all rows set `maxrows` to zero:

```
maxrows 0
```

To return more than 100 rows set `maxrows` to a higher value. For example, to return 500 rows:

```
maxrows 500
```

Note: If running the Interactive SQL console from a shortcut, you must modify the shortcut **Target** to include the `-maxrows` parameter with the desired value.



The ODBC driver embeds a third party SQL/ODBC engine from DataDirect, a division of Progress Software.

See the [DataDirect SQL Reference](#) for information on proper SQL syntax.

Note: The ServiceNow ODBC driver only supports SELECT statements. The driver ignores other SQL statements such as CREATE and ALTER.

Use the ServiceNow ODBC driver in SQL Server as a Linked Server.

Using the ODBC driver in SQL Server as a [Linked Server](#) allows SQL Server to query tables from a ServiceNow instance directly via the ODBC driver. Only use the procedures described with SQL Server 2008 and 2012. Other versions of SQL Server may cause unexpected behavior. If you encounter unexpected behavior, refer to the [troubleshooting linked server Knowledge Base article](#).

Required Permissions

Additional information on the required permissions for SQL Server Linked Servers can be found [on the MSDN blog](#).

Note: Review this information if you encounter permission errors with SQL Server.

- [ODBC SQL Server video tutorials](#)

Watch video tutorials about configuring and troubleshooting the ODBC driver with a SQL Linked Server.

- [Configure SQL Server](#)

The following example configuration was performed on SQL Server 2008, installed on Windows Server 2008.

Watch video tutorials about configuring and troubleshooting the ODBC driver with a SQL Linked Server.

Configuring Microsoft SQL Linked Server with the ODBC driver

Troubleshooting Microsoft SQL Linked Server permissions

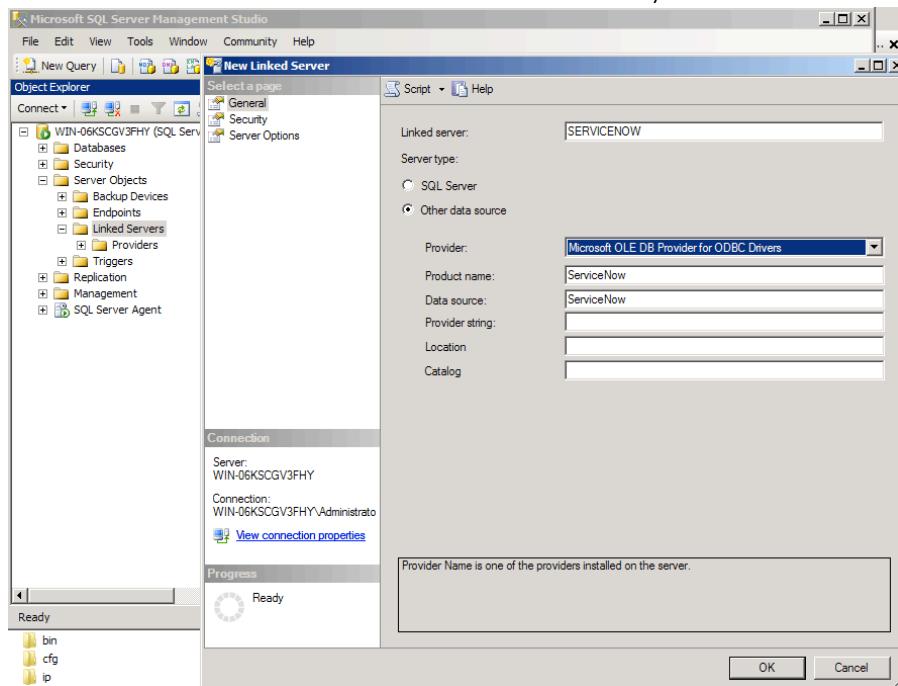
The following example configuration was performed on SQL Server 2008, installed on Windows Server 2008.

Before you begin

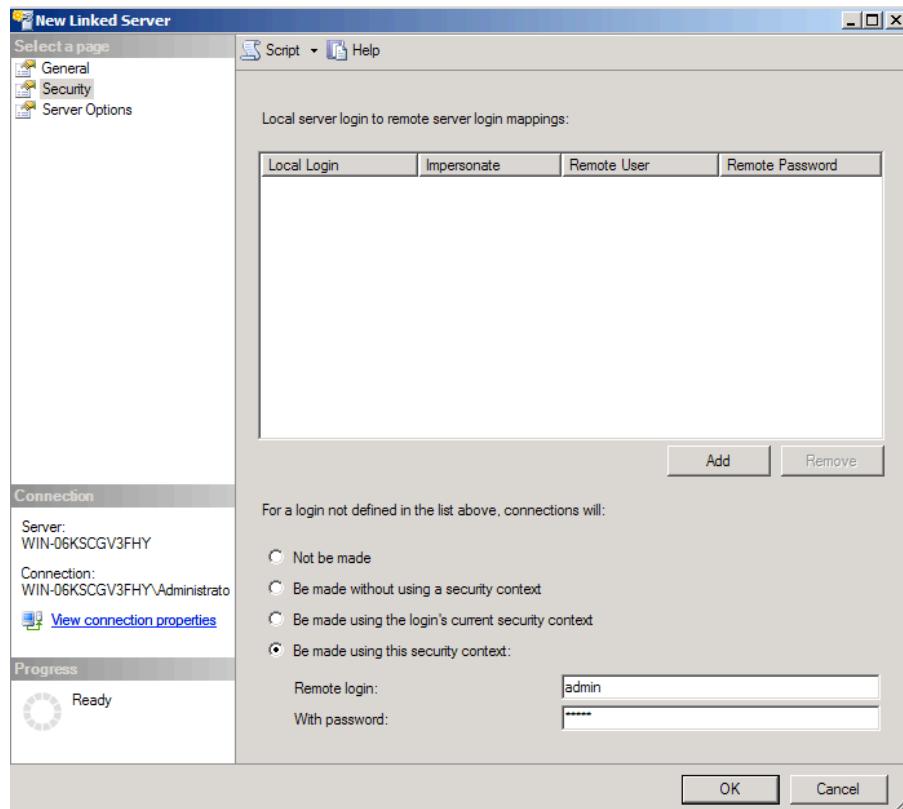
The ODBC driver must be installed on the same computer on which SQL Server is installed.

Procedure

1. Right-click the SQL Server Management Studio application and select **Run as Administrator**.
2. Log in to the database to which you want to link.
3. Right-click **Server Objects > Linked Servers**.
4. Click **New Linked Server**.
5. Enter the following values in the dialog.
 - **Linked server:** SERVICENOW. This is the name of the Linked Server.
 - **Provider:** Microsoft OLE DB Provider for ODBC drivers
 - **Product name:** ServiceNow. This is an identifier. Enter any value that is appropriate.
 - **Data source:** ServiceNow. This is the name of your DSN.



6. Select **Security** from the Select a page list, and then enter the following security values:
 - a. For a login connection, select **Be made using this security context**.
 - b. Enter the user name and password for connecting to the ServiceNow instance.
 - c. Click **OK**.

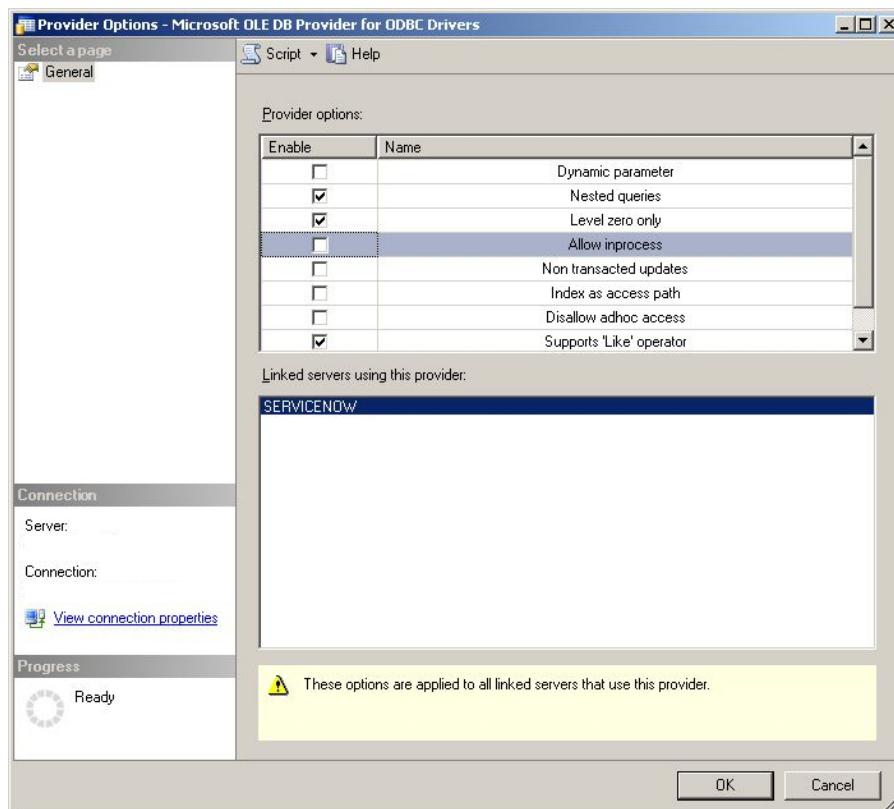


7. Navigate to **Server Objects > Linked Server > Providers** and double-click Microsoft **OLE DB Provider for ODBC drivers**.
8. Select the following options.
 - Nested Queries
 - Level zero only

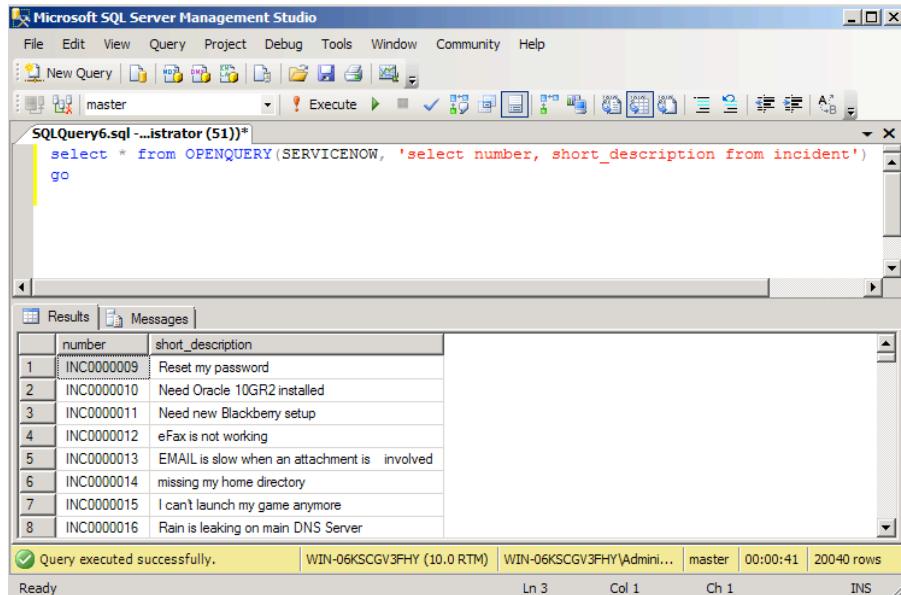
- Support 'Like' operator

Tip:

ServiceNow recommends running the third-party provider in the out-of-process mode setting (AllowInProcess=FALSE). If you run the provider in-process (within the same process as SQL Server), then any issues with the provider can affect the SQL Server process, which in turn could result in crashing SQL server.



9. Test your connection by selecting the newly created linked server SERVICENOW and selecting **Test connection**.
10. Execute the following query in a query builder window to retrieve some results.



What to do next

Number Precision Errors

You may encounter precision errors querying for decimal or number field values using the OPENQUERY syntax with the ODBC driver. In this case, use the Cast syntax to convert the precision. For example:

```
SELECT * FROM OPENQUERY (SERVICENOW , 'select Cast(sys_mod
_count as Decimal(38,0)), number, short_description from i
ncident' ) GO
```

SQL Server Connection String

To use the ODBC driver directly in SQL Server 2008, specify the connection string in the following format.

```
Dsn=ServiceNow;uid =username;pwd =password
```

Note: The latest SQL Server 2008 patches are required for the ability to specify a connection string in the user interface, via the SQL import wizard

Using sp_addlinkedserver

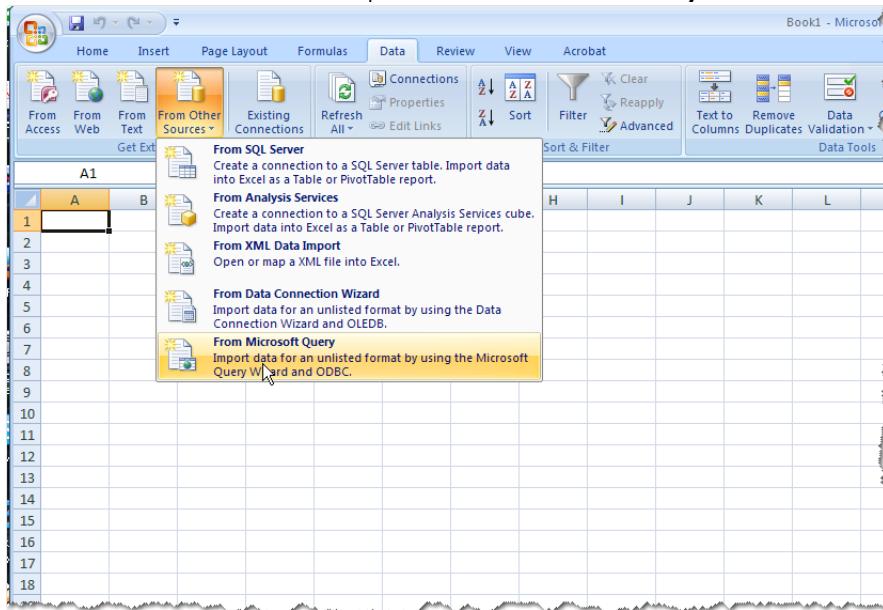
The following example creates a linked server named "ServiceNow ODBC" that uses the Microsoft OLE DB Provider for ODBC (MSDASQL) and the data_source parameter

```
EXEC sp_addlinkedserver
    @server = N 'ServiceNow ODBC' ,
    @srvproduct = N '',
    @provider = N 'MSDASQL' ,
    @datasrc = N 'ServiceNow';
GO
```

After creating the linked server, you must update its properties to specify the login credentials.

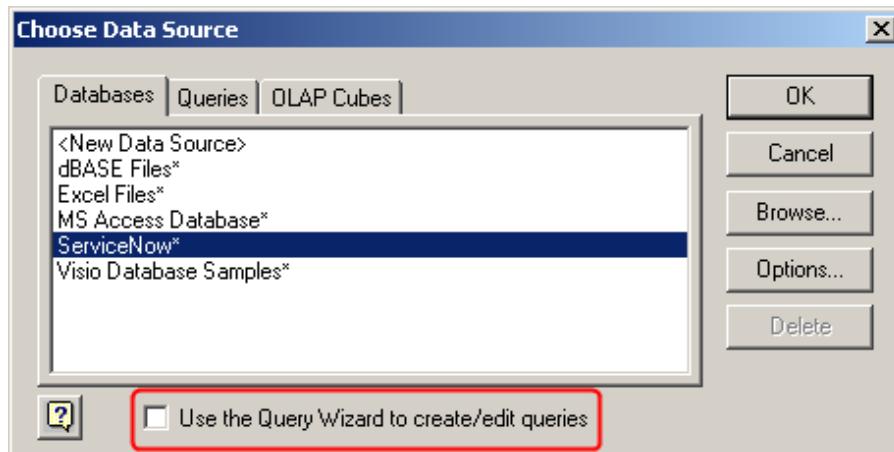
After installing the ODBC driver and its associated DSN, use it in Excel as a data source provider.

1. In Excel open the **Data** tab.
2. Under **From Other Sources** open **From Microsoft Query**.

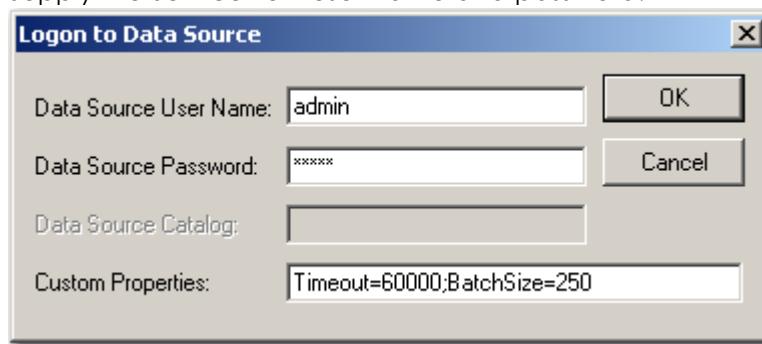


3. Select **ServiceNow** as your database (the default DSN name).
4. Clear the **Use the Query Wizard to create/edit queries** check box.

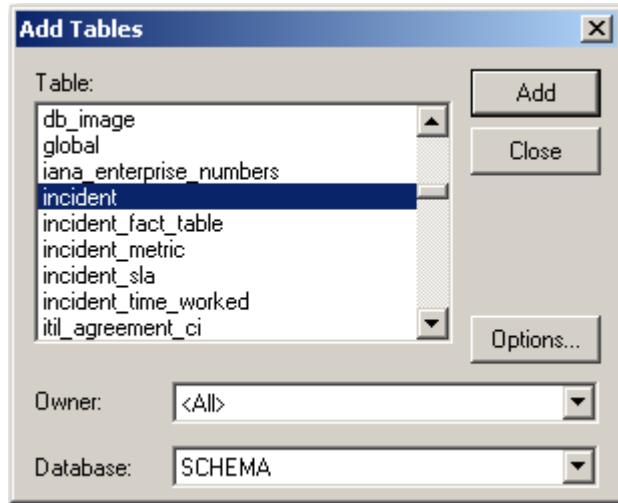
Note: The Excel Query Wizard does not support the listing of columns from a table name that contain an underscore (_). Clearing this check box uses the Query Builder instead, which supports the use of this character.



- Supply the ServiceNow user name and password.



- Select a table from the ServiceNow instance and click **Add**.



7. Close the dialog box.
8. Select the table columns from which the Query Builder will retrieve data. Use the list above the table, or type the names directly into the columns, and then press **Enter**.
9. To retrieve the data and create the Excel record, click the **Return Data** icon or select **File > Return Data to Microsoft Office Excel**.

Microsoft Query

File Edit View Format Table Criteria Records Window Help

SQL

Query from local

incident

number	short_description	dy_assigned_to	dy_state	state
INC0000009	Reset my password	David Loo	Open	1
INC0000010	Need Oracle 10GR2 installed	Don Goodlife	Open	1
INC0000011	Need new Blackberry setup	ITIL User	Open	1
INC0000012	fax is not working	David Loo	Open	1
INC0000013	EMAIL is slow when an attach	David Loo	Open	1
INC0000014	missing my home directory	ITIL User	Open	1
INC0000015	I can't launch my game anymore	Don Goodlife	Open	1
INC0000016	Rain is leaking on main DNS S	ITIL User	Open	1
INC0000017	How do I create a sub-folder	David Loo	Open	1
INC0000018	Sales forecast spreadsheet is	ITIL User	Open	1
INC0000019	Can't launch X-Win32	Bud Richman	Open	1
INC0000020	Request for a Blackberry	ITIL User	Open	1
INC0000021	New employee hire	Beth Anglin	Open	1
INC0000024	Issue with a web page	ITIL User	Open	1
INC0000025	I need more memory	ITIL User	Open	1
INC0000026	Seem to have an issue with my	Don Goodlife	Open	1
INC0000028	My disk is still having issues. C	Don Goodlife	Open	1
INC0000027	please remove this hostfile	ITIL User	Open	1
INC0000029	I cant get my weather report	Don Goodlife	Open	1
INC0000030	Lost connection to the wireless	David Loo	Open	1
INC0000031	EMAIL Server Down	David Loo	Open	1
INC0000032	EMAIL Server Down Again	David Loo	Open	1
INC0000033	File Server is 80% full - Needs	Don Goodlife	Open	1
INC0000034	Does not look like a backup o	David Loo	Open	1
INC0000035	Reset my password	Luke Wilson	Open	1
INC0000036	Issue with networking	Luke Wilson	Open	1
INC0000037	Request for a new service	Howard Johnson	Open	1

The requested data is brought into Excel.

A	B	C	D	E
number	short description	dv_assigned_to	dv_state	state
1	INC0000009 Reset my password	David Loo	Open	1
2	INC0000010 Need Oracle 10GR2 installed	Don Goodliffe	Open	1
3	INC0000011 Need new Blackberry setup	ITIL User	Open	1
5	INC0000012 eFax is not working	David Loo	Open	1
6	INC0000013 EMAIL is slow when an attachment isinvolved	David Loo	Open	1
7	INC0000014 missing my home directory	ITIL User	Open	1
8	INC0000015 I can't launch my game anymore	Don Goodliffe	Open	1
9	INC0000016 Rain is leaking on main DNS Server	ITIL User	Open	1
10	INC0000017 How do I create a sub-folder	David Loo	Open	1
11	INC0000018 Sales forecast spreadsheet is READ ONLY	ITIL User	Open	1
12	INC0000019 Can't launch X-Win32	Bud Richman	Open	1
13	INC0000020 Request for a Blackberry	ITIL User	Open	1
14	INC0000021 New employee hire	Beth Anglin	Open	1
15	INC0000024 Issue with a web page	ITIL User	Open	1
16	INC0000025 I need more memory	ITIL User	Open	1
17	INC0000026 Seem to have an issue with my harddrive...	Don Goodliffe	Open	1
18	INC0000028 My disk is still having issues. Can't delete afile	Don Goodliffe	Open	1
19	INC0000027 please remove this hotfix	ITIL User	Open	1
20	INC0000029 I cant get my weather report	Don Goodliffe	Open	1
21	INC0000030 Lost connection to the wirelessnetwork	David Loo	Open	1
22	INC0000031 EMAIL Server Down	David Loo	Open	1
23	INC0000032 FMAIL Server Down Again	David Loo	Open	1

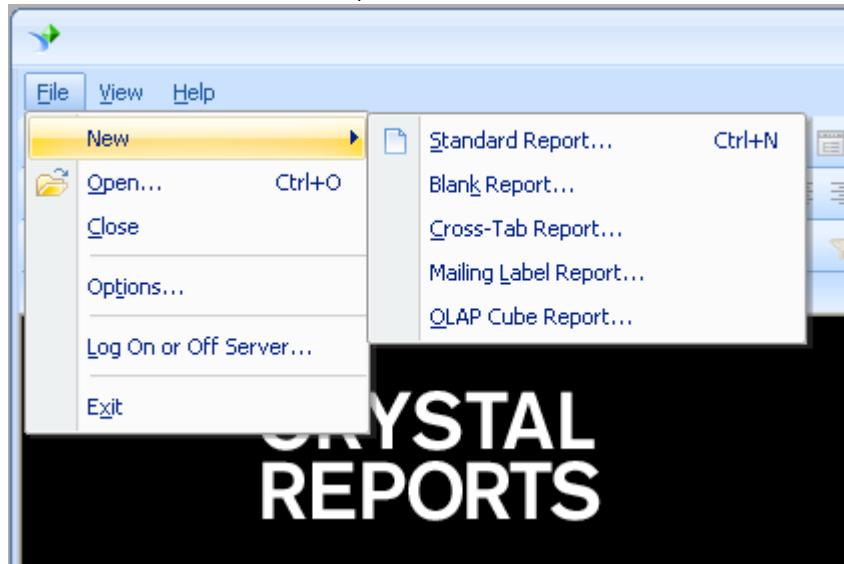
After installing the ODBC driver and its associated DSN, use it in Crystal Reports as a data source provider.

About this task

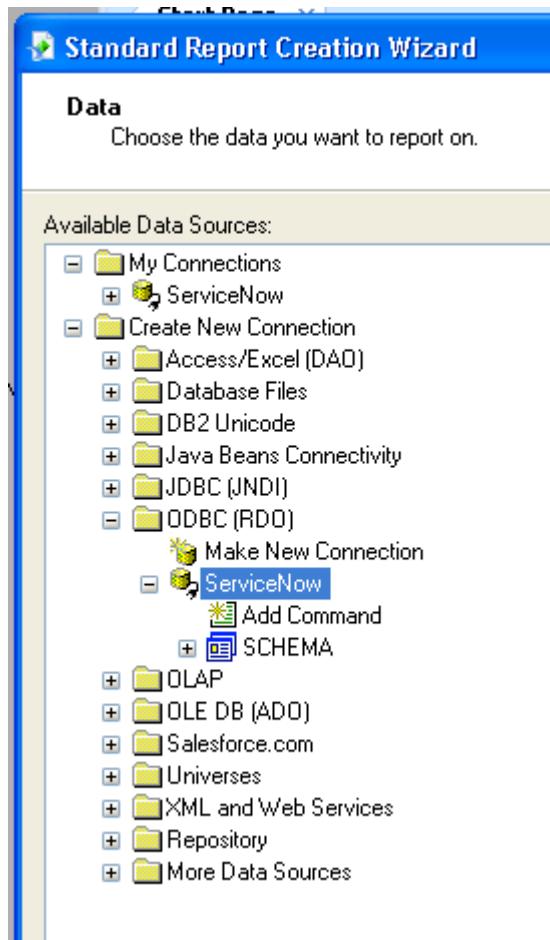
Note: Crystal Reports includes the configuration file CRConfig.xml that contains the JVM minimum heap size (Xms) and maximum heap size (Xmx) values. When configuring the ODBC driver with Crystal Reports, ensure that the ODBC driver uses the same minimum and maximum JVM heap size as Crystal Reports. If these values do not match, update the ODBC driver settings, not the Crystal Reports settings.

Procedure

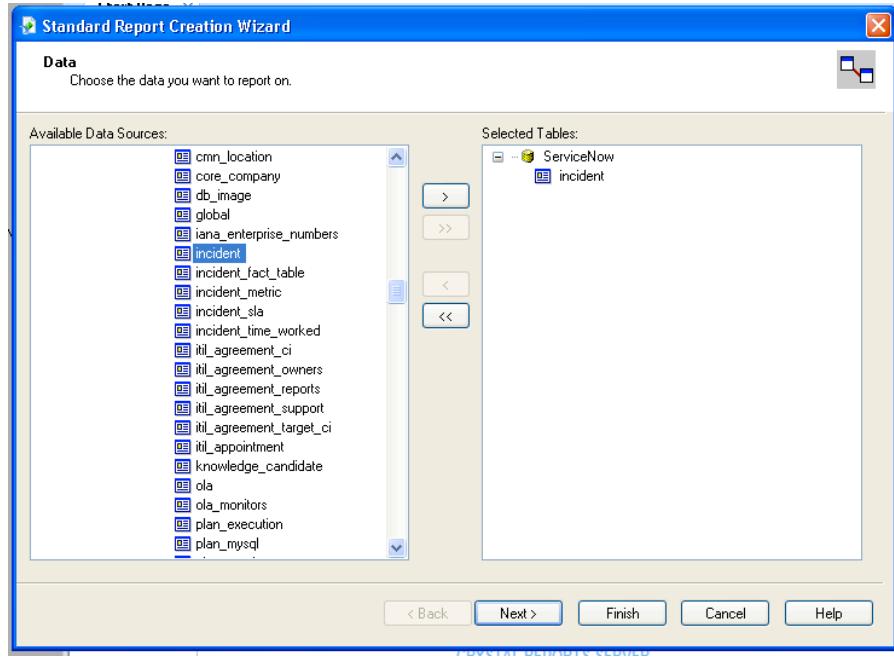
1. Create a new Standard Report.



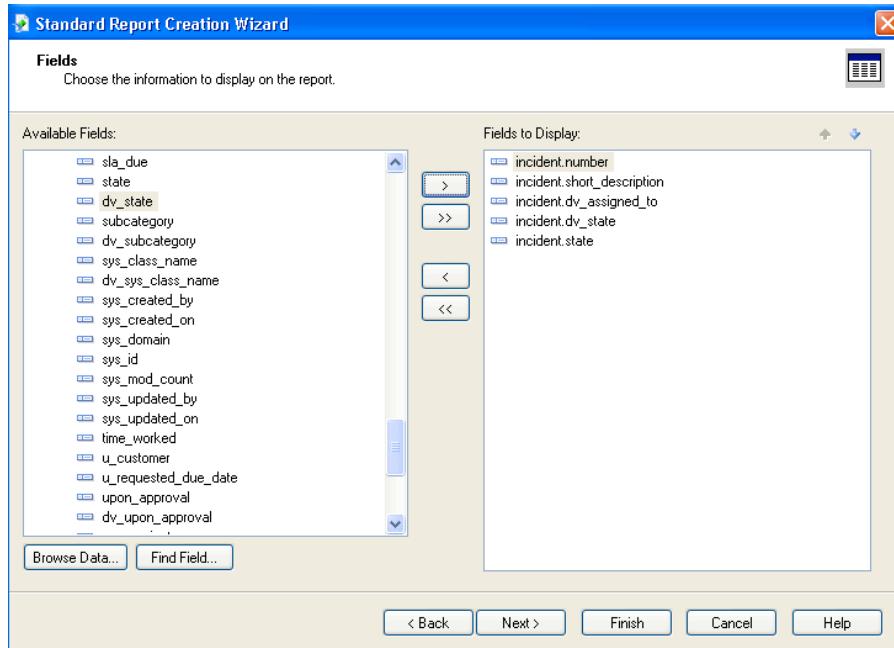
2. Create a new connection using the ServiceNow DSN.



3. Select a table from the list of available tables.



4. Select the available fields from the selected table.



5. Click **Finish** to render the report.

number	short_description	dv_assigned_to	dv_state	state
INC000009	Reset my password	David Loo	Open	1
INC000010	Need Oracle 10GR2 installed	Don Goodlife	Open	1
INC000011	Need new Blackberry setup	ITIL User	Open	1
INC000012	grfx is not working	David Loo	Open	1
INC000013	EMAIL Server down again	David Loo	Open	1
INC000014	missing my home directory		Open	1
INC000015	I can't launch my game anym		Open	1
INC000016	Rain is leaking on main DNS		Open	1
INC000017	How do I create a sub-folder		Open	1
INC000018	Sales forecast spreadsheet		Open	1
INC000019	Can't launch X-Win32		Open	1
INC000020	Request for a Blackberry		Open	1
INC000021	New employee hire...	Beth Anglin	Open	1
INC000024	Issue with a web page	ITIL User	Open	1
INC000025	I need more memory		Open	1
INC000026	Seem to have an issue with r	Don Goodlife	Open	1
INC000028	My disk is still having issues.	Don Goodlife	Open	1
INC000027	please remove this host	System Administrator	Open	1
INC000029	I cant get my weather report		Open	1
INC000030	Lost connection to the wirele	David Loo	Open	1
INC000031	EMAIL Server Down	David Loo	Open	1
INC000032				

Domain separation and ODBC driver

This is an overview of domain separation and ODBC drivers. Domain separation enables you to separate data, processes, and administrative tasks into logical groupings called domains. You can control several aspects of this separation, including which users can see and access data.

Support level: Basic*

- There is business logic to ensure data goes into the proper domain for the application's service provider use cases.
- In the application, the user interface, cache keys, reporting, rollups, aggregations, and so on, all consider domain at run time.
- The owner of the instance needs to be able to set up the application to function normally across multiple tenants.

Use case: As a service provider when I use chat to respond to a tenant-customer's message, the client must be able to see my response.

Understanding domain separation and web services

Domain separation is supported in Web Services. Domain separation enables you to separate data, processes, and administrative tasks into logical groupings called domains. You can control several aspects of this separation, including which users can see and access data.

Support level: Standard*

The support level is Standard but has some exceptions or special conditions.

- Includes **Basic** level
- Business logic: The service provider (SP) creates or modifies processes per customer. The use cases reflect proper use of the application by multiple SP customers in a single instance.
- The instance owner must be able to configure minimum viable product (MVP) business logic and data parameters. This configuration is done per tenant, as expected for the specific application.

Sample use case: An admin must be able to make comments required when a record closes for one tenant, but not for another.

For more information on support levels, see [Application support for domain separation](#).

Analyze REST and SOAP API usage

REST and SOAP API analytics allow you to track and analyze web service API usage.

Use analytics to help answer questions such as

- Which APIs are used most?
- Which API versions are being used? Can I deprecate older versions?
- Which API methods are being used?
- What resources are being accessed?
- Who is using each API and resource?

The instance tracks analytics for all inbound web services, including platform web services such as the REST Table API or the SOAP API, and custom web services such as scripted REST APIs and scripted SOAP web services. Analytics are aggregated by each resource and HTTP action combination.

Outbound web services, such as REST Messages, are not tracked.

- [Collect analytics for an API](#)

Select which APIs to collect analytics for and select if requestor information should be collected for each API.

- [REST & SOAP API analytics dashboards](#)

You can view API analytics for the entire instance or for specific APIs using the included dashboards.

- [REST and SOAP API analytics collection and cleanup](#)

API analytics uses scheduled jobs to collect and clean up analytics data.

- [API analytics properties](#)

Certain properties control the behavior of API analytics.

Collect analytics for an API

Select which APIs to collect analytics for and select if requestor information should be collected for each API.

Before you begin

Note: API analytics is not available for certain APIs used for internal platform functionality, such as the UI and Mobile app APIs.

Role required: `api_analytics_read` or `admin`

About this task

When a new API is added, an inclusion list record is created automatically. You can modify or create new inclusion list records to manually configure which APIs and requestor information are logged.

Procedure

Navigate to **All > System Web Services > Inclusion List** and create a new record.

API Transactions Stats Inclusion Lists fields

Field	Description
API name	For REST APIs, enter the API name. This name should contain only the namespace and ID of the API. Do not include any request parameters in the API name. For example, for the Table API endpoints api/now/table/incident and api/now/table/problem, the namespace and ID are now/table .
Collect API stats	SOAP APIs do not support separate logging configurations for different APIs. You can configure logging for all SOAP APIs by modifying the SOAP APIs inclusion list record.
Collect API requestor stats	Select this check box to track analytics for the specified API.

REST & SOAP API analytics dashboards

You can view API analytics for the entire instance or for specific APIs using the included dashboards.

REST & SOAP API Analytics includes dashboards that present the overall analytics, analytics per API, and analytics per requesting user.

To access API Analytics dashboards, navigate to **System Web Services** and select **Usage Overview**, **Usage by API**, or **Usage by Requestor**. Users with the `api_analytics_read` or `admin` roles can view these dashboards.

Note: The Usage by Requestor dashboard uses the Responsive Canvas feature available with Istanbul. For the best experience on upgraded instances, you may need to [enable responsive dashboards](#).

The following dashboards are available:

Available API analytics dashboards

Dashboard	Description
Usage Overview	Provides general usage statistics for all REST and SOAP APIs.
Usage by Web API	Provides detailed usage statistics for each API.
Usage by Requestor	Provides detailed usage statistics for each requesting user. You can additionally filter by API to view detailed stats for a specific user and API combination. If the selected user has not made requests to the selected API, no data is shown for that combination.

Dashboard	Description
	<p>Note: To view API stats for all users, do not clear the user breakdown selection. Instead, use the Usage by Web API dashboard.</p>

You can access API analytics from the REST API Explorer. When exploring



an API, click the context menu icon () then select **API analytics** to view analytics for that specific API.

For custom web services, such as scripted REST APIs or scripted SOAP web services, you can access analytics for the API by clicking the **API analytics** related link on the scripted REST Service or scripted Web Service forms.

When you directly access the analytics for an API, if there are no analytics for that specific API, the analytics for all APIs appears.

Related concepts

- [Use the REST API Explorer](#)

REST and SOAP API analytics collection and cleanup

API analytics uses scheduled jobs to collect and clean up analytics data.

The instance tracks all web service transactions for APIs on the inclusion list and maintains a daily history, aggregated by resource and HTTP action combination. Requester information is aggregated per requester, resource, and HTTP action combination and tracked up to the daily limit defined by the property com.glide.api.stats.daily_limit.

Refer to the following table to determine which requests are logged.

Logged requests

Whitelist API name	Example resource	Response code	Description	Logged
now/table	/api/now/table/incident	Any except 401	Valid resource and table	Yes
now/table	/api/now/table/invalidResource	400	Valid resource but an invalid table	Yes
now/table	/api/now/table/incident	403	Requesting user has insufficient privileges	Yes
now/table	/api/now/table/incident	401	Requesting user is not authenticated	No
myApp/myScriptedApi	myApp/myScriptedApi/myResource	Any except 401	Valid resource	Yes
myApp/invalidApiName	<ul style="list-style-type: none"> • myApp/invalidApiName • myApp/invalidApiName/myResource 	400	Invalid API, even with a matching inclusion list entry	No

On the 2nd of each month, the API Monthly Stats scheduled job calculates the monthly total for each resource and HTTP action

combination. Each day the API Monthly Requestor Stats scheduled job calculates the monthly total for each resource, requester, and HTTP action combination based on daily scores older than 2 days.

Daily statistics are maintained for 33 days. Monthly totals are maintained for 13 months. Table cleaners for the sys_api_stats, sys_api_stats_requestor, and sys_api_stats_requestor_monthly tables remove analytics records older than these limits.

REST & SOAP API analytics naming

The **API Name** used when tracking API analytics is determined by the type of API being described, such as a REST API or a Scripted SOAP service.

API type	Description
REST	The API namespace and the first part of the URI following the namespace is used as the API name. For example, for the Table API endpoints api/now/table/incident and api/now/table/problem, the namespace and ID are now/table .
Direct SOAP (table does not extend Import Set Row table)	If the direct SOAP request accesses a table, Direct SOAP is used as the API name.
SOAP import (table extends Import Set Row table)	Import Set SOAP is used as the API name.
Scripted SOAP Services	The SOAP request endpoint page is used as the API name, such as my_service.do.

API analytics properties

Certain properties control the behavior of API analytics.

Property	Description
com.glide.api.stats.enabled	<p>When true, enables the collection of API usage statistics. When false, no analytics are collected even for domains on the inclusion list.</p> <ul style="list-style-type: none">• Type: true false• Default value: true• Location: Add to the System Properties [sys_properties] table
com.glide.api.stats.max_bytes_nursery_size	<p>The maximum amount of memory, in bytes, used to store transaction data before it is written to the log. Transaction data is written to the log regularly based on the value of com.glide.api.stats.persist_interval.</p> <p>If a large volume of transactions exceeds this memory limit before the log is written, some transactions may not be logged. The event api.stats.cache.size.reached is fired if this limit is reached.</p> <p>This property value must be between 1 and 3 megabytes.</p> <ul style="list-style-type: none">• Type: integer• Default value: 3145728

Property	Description
	<ul style="list-style-type: none">• Location: Add to the System Properties [sys_properties] table
com.glide.api.stats.persist_interval	<p>The frequency, in seconds, for writing transactions stored in memory to the log. This property value must be between 30 and 120 seconds.</p> <ul style="list-style-type: none">• Type: integer• Default value: 60• Location: Add to the System Properties [sys_properties] table
com.glide.api.stats.daily_limit	<p>The daily limit of requestor stats records per instance node. As soon as the value is reached, data is no longer aggregated and stored for that day. The event api.stats.requestor.daily.limit.reached is fired if this limit is reached.</p> <ul style="list-style-type: none">• Type: integer• Default value: 20000• Location: Add to the System Properties [sys_properties] table
glide.api.stats.debug	<p>When true, enables debug logging for API stats.</p> <ul style="list-style-type: none">• Type: true false• Default value: false

Property	Description
	<ul style="list-style-type: none">Location: Add to the System Properties [sys_properties] table

Query record data using the GraphQL API framework

Create a custom GraphQL API to query record data from a component or a third-party system.

For example, you can create a component that displays the cases associated with an SLA. You can use the Next Experience UI Framework to develop the component you need, and access case data from the platform by creating a GraphQL schema that defines data in the Case table.

To learn more about developing components, see [Developing components for Workspace](#).

Benefits of GraphQL

GraphQL is a web query language optimized for client-side development. Using scripted GraphQL, you can:

- Discover fields and objects available to query through introspection.
- Query the exact data you need from a component.
- Manage multiple possible queries from a single API, as opposed to multiple endpoints for a REST request.
- Integrate with third-party systems by making the schema public.
- Generate the GraphQL query from your component and handle the response.

What to know before you begin

Before you start creating custom GraphQL APIs, make sure you have:

- GraphQL knowledge to create a schema.
- JavaScript knowledge to define the API behavior.

- General knowledge of web component concepts.
- A custom Workspace component to consume record data.
- Understanding of the ServiceNow data model that you want to expose in the schema.
- GlideRecord knowledge to map fields to record data in your resolver scripts.

GraphQL overview

Creating a scripted GraphQL API includes these parts:

GraphQL Schema Definition Language (SDL)

Define the structure and data type of fields available in a GraphQL query. You can define the SDL using the **Schema** script field in the GraphQL Scripted Schemas [sys_graphql_schema] table. The SDL only supports Query and Mutation operations.

Resolvers

Define the data returned by each field. You can define the resolvers for each field in the GraphQL Scripted Resolvers related list on the GraphQL Scripted Schemas form.

Type resolvers

Resolve interfaces and unions into concrete GraphQL types. For example, you might define a union between an incident type and a problem type. Use the typeresolver script to define when to return which. You can define the typeresolvers in the GraphQL Scripted Typeresolvers related list on the GraphQL Scripted Schemas form.

Resolver mappings

Map resolvers to fields in the schema. You can define resolver mappings in the GraphQL Scripted Resolver Mappings related list on the GraphQL Scripted Schemas form.

To learn more about the GraphQL query language, see the [GraphQL website](#).

Limitations

The following GraphQL features aren't supported:

- Subscription operations
- Custom scalar types

Introspection

By default, introspective queries into your custom schemas aren't allowed. To allow introspection, set a system property. See [GraphQL system properties](#).

Namespaces

GraphQL APIs have two different namespaces:

Application namespace

The namespace for the custom application. To learn more about application namespaces, see [Application scope](#).

Schema namespace

The namespace for the schema to ensure that all queries are unique. You can have multiple schema namespaces in a single application.

When querying data, you must include both namespaces in your query. For example, the following query is searching for data with the following namespaces:

- Application namespace: `x_graph_scope`
- Schema namespace: `planet`

```
query {  
  x_graph_scope {  
    planet {  
      findAll {  
        name  
        mass  
        distance  
      }  
    }  
  }  
}
```

```
}
```

Directives and global functions

@source schema directive

Maps a GraphQL field to the value of a property of the parent object. If the field has a separate resolver script, the system uses the record that it resolves to instead of the parent object.

Use the `@source` directive in your schema script.

@defer query directive

Defer processing of a GraphQL fragment until later in the query. Use this query directive to delay returning data for slow-responding fields within a fragment. Stream the field results of the deferred fragment as a multi-part response.

Note: To use the `@defer` directive, your GraphQL client must accept multipart/mixed HTTP headers. For example, set the HTTP headers to `Accept: multipart/mixed; boundary="-"`.

Use the `@defer` directive to improve user time to interaction. Avoid applying this query directive indiscriminately as it can also cause performance degradations. Conduct performance testing to determine which fields can be deferred for better performance.

Resolver functions

These functions are available on the global env object.

- `getArguments()`: Returns the arguments of the previous field.
- `getSource()`: Returns the parent object.

Use in your resolver script.

Typeresolver functions

These functions are available on the global env object.

- `getArguments()`: Returns the arguments of the previous field.
- `getObject()`: Returns the parent object.
- `getTypeName()`: Returns the name of the interface or union type.

Use in your typeresolver script.

Demo application

To see a demo GraphQL PTO calendar schema with mutations and queries, enable the GraphQL Framework Demo Application plugin (com.glide.graphql.framework.demo).

- [Create a GraphQL schema](#)

Create a GraphQL schema to make data available to GraphQL queries.

- [Test a GraphQL schema](#)

Test your GraphQL schema using a third-party tool.

- [Query a GraphQL schema from a component](#)

Access record data in a component by querying your scripted GraphQL schema.

- [GraphQL system properties](#)

Configure GraphQL API framework behavior. For example, you can configure whether to allow introspective queries into your schema.

Create a GraphQL schema

Create a GraphQL schema to make data available to GraphQL queries.

Before you begin

Role required: admin

Procedure

1. Navigate to **All > System Web Services > GraphQL > GraphQL APIs**.

2. Click **New**.
3. Complete these fields.

Field	Description
Name	Name of the schema.
Schema namespace	Must be unique within the name of the application.
Application	Read-only application that the schema is a part of.
Application namespace	Read-only. Works with the Schema namespace to prevent conflict with schemas with the same name.
Active	Checked if active, unchecked if not active.
Schema	Schema definition that adheres to the GraphQL SDL format. Must be GraphQL valid syntax. Otherwise, error messages appear on save to indicate the syntax problem. Note: The following GraphQL features are not supported: <ul style="list-style-type: none">• Subscription operations• Custom scalar types

You can use this directive in your schema:

`@source`: Maps a GraphQL field to the value of a property of the parent object. If the field has a separate resolver script, the system uses the record that it resolves to instead of the parent object.

This example defines an Incident object type in the schema and uses a resolver script to map the type to a GlideRecord from the Incident table. Using the `@source` directive maps the fields within the Incident type to the `value` or `display_value` in the Incident GlideRecord.

```
type Incident {  
    id: String @source(value: "sys_id.value")  
    active: Boolean @source(value: "active.display_value")  
    state: String @source(value: "state.display_value")  
    priority: String @source(value: "priority.display_value")  
    severity: String  
    description: DisplayableString  
    resolvedBy: User @source(value: "resolved_by.value")  
    openedBy: User @source(value: "opened_by.value")  
    child_incidents: String  
    opened_at: String  
    resolved_at: String  
    closed_at: String  
    work_notes: String  
    comments: String @source(value: "comments.display_value")  
    parent_incident: String  
}
```

4. Select the **Security** tab and complete the form.

Field	Description
Requires authentication	Checked if the schema requires authentication, unchecked if the schema doesn't require authentication and is available publicly.
Requires SNC internal	Appears only if the Explicit Roles plugin is enabled. Checked if the schema requires the SNC internal role, unchecked if the schema doesn't require the SNC internal role.
Requires ACL authorization	Checked if the schema requires ACL authorization, unchecked if the schema doesn't require ACL authorization.

Field	Description
ACLs	Appears only if Requires ACL authorization is checked. Checks incoming queries against ACLs of type GraphQL .

Note: The API returns clear error messages to users who do not have access to a schema, or who are not authenticated when the API requires authentication to access.

5. Save the form.

6. (Optional) Create a resolver script to define what value the schema returns when a component queries a specific field.

If you don't define a resolver for a field, the query returns any matching field value from the parent object type. For example, suppose you have an Incident object type in your schema with a worknotes field. The Incident object type has a resolver that maps to a GlideRecord from the Incident table. If you do not create a resolver mapping for the worknotes field, the system searches the parent object's data source, which is the GlideRecord from the Incident table, for a worknotes field and assigns the associated value.

a. Select the GraphQL Scripted Resolvers tab and click **New**.

b. Complete the form.

Field	Description
Name	Name of the resolver.
Schema	Read-only schema namespace.
Application	Read-only application that the schema is a part of.
Script	Define the value returned when the field is queried. Functions available on the global env object: <ul style="list-style-type: none">• <code>getArguments()</code>: Returns the arguments of the previous field.

Field	Description
	<ul style="list-style-type: none">• <code>getSource()</code>: Returns the parent object. <p>This script has access to GlideRecord.</p>

This example returns a record from the User table when the associated field is queried:

```
(function process /*ResolverEnvironment*/ env) {  
  
    var userid = env.getArguments().id != null ? env  
        .getArguments().id : env.getSource();  
    var now_GR = new GlideRecord('sys_user');  
    gr.addQuery('sys_id', userid);  
    gr.query();  
    return gr;  
  
}) (env);
```

c. Click **Submit**.

7. (Optional) Define the typeresolvers for your schema to resolve interfaces and unions into concrete types.
 - a. Select the GraphQL Scripted Typeresolvers tab and click **New**.
 - b. Complete the form.

Field	Description
Schema	Read-only schema namespace.
Type	The interface or union type defined in the schema.
Application	Read only application that the schema is a part of.
Script	Define the value returned for union and interface types. Functions available on the global env object:

Field	Description
	<ul style="list-style-type: none">• <code>getArguments()</code>: Returns the arguments of the previous field.• <code>getObject()</code>: Returns the parent object.• <code>getTypeName()</code>: Returns the name of the interface or union type. <p>This script has access to <code>GlideRecord</code>.</p>

- c. Click **Submit**.
8. (Optional) Map the resolver and typeresolver records to fields in the schema. This mapping lets the system know what value to return when a component queries a specific field.
- Select the GraphQL Scripted Resolver Mappings tab and click **New**.
 - Complete the form.

Field	Description
Path	Path to the field in the schema you want to map.
Resolver	Resolver to use to define the data returned by the field.
Application	Read-only application that the schema is a part of.
Schema	Read-only schema namespace.

- c. Click **Submit**.

Test a GraphQL schema

Test your GraphQL schema using a third-party tool.

Before you begin

Role required: admin

- [Create a GraphQL schema](#).
- Download a third-party GraphQL tool, such as GraphiQL or Insomnia.

Procedure

1. Open the third-party tool on your machine.
2. Set the endpoint to query your custom schema: `https://<your-instance>.servicenow.com/api/now/graphql`.
3. Add any authentication headers required.
4. (Optional) If introspection is enabled, you can send introspective queries to learn more about what the schema includes. To turn on introspection, see [GraphQL system properties](#).
Note: Do not use introspective queries in a production environment.
5. Send test queries to the schema.

Include applicable namespaces

Include both the application and schema namespaces in the query.

Defer slow-responding fields

You can apply the `@defer` query directive to a fragment to delay the processing of one or more slow-responding fields. Deferring slow-responding fields allows the query to return other fields first.

Warning: Avoid applying this query directive indiscriminately as it can also cause performance degradations. Conduct performance testing to determine which fields can be deferred for better performance.

This example shows using both the application and schema namespaces:

- Application namespace: `x_graph_scope`
- Schema namespace: `planet`

```
query {
  x_graph_scope {
    planet {
      findAll {
        name
        mass
        distance
      }
    }
  }
}
```

This example shows using the `@defer` directive to stream results for the `openedBy` field after the initial incident response.

```
query findIncidents {
  now {
    incident {
      findAll {
        description {
          displayValue
        }
        ...
        @defer(label: "my-label", if: true) {
          openedBy {
            firstName
            lastName
          }
        }
      }
    }
  }
}
```

What to do next

[Query a GraphQL schema from a component.](#)

Query a GraphQL schema from a component

Access record data in a component by querying your scripted GraphQL schema.

Before you begin

Role required: admin

1. [Create a GraphQL schema](#).
2. [Test a GraphQL schema](#).
3. [Disable introspective queries in your production environment](#).
4. [Create a custom component](#).

Procedure

1. Navigate to your component directory.
2. Define the query in your component's index.js file.

Remove introspective queries

Do not use introspective queries for components deployed in a production environment.

Include applicable namespaces

Include both the application and schema namespaces in the query.

Defer slow-responding fields

You can apply the `@defer` query directive to a fragment to delay the processing of one or more slow-responding fields. Deferring slow-responding fields allows the query to return other fields first.

Warning: Avoid applying this query directive indiscriminately as it can also cause performance degradations. Conduct performance testing to determine which fields can be deferred for better performance.

Note:

```
const incidentQuery = `query ($sys_id: String!) {
  query (sys_id: $sys_id) {
    incident {
      _results {
        sys_id {
          displayValue
        }
      }
    }
  }
}`;
```

3. Add the following line to the top of the script to import the `createGraphQLEffect` API into your component.

```
import {createGraphQLEffect} from '@servicenow/ui-effect-graphql';
```

4. Call the query using the `createGraphQLEffect` API. To learn more about this API, see the [Developer Site](#).
5. Create action handlers to manage the result of the query.

```
actionHandlers: {
  [DATA_FETCH_REQUESTED]: dataFetchHandler,
  [DATA_FETCH_STARTED]: ({updateState}) => updateState({loading: true}),
  [DATA_FETCH_SUCCEEDED]: ({action, updateState}) => updateState({data: action.payload}),
  [DATA_FETCH_FAILED]: () => { /* handle network error */ },
}
```

Example

This example shows a complete component including the query, the GraphQL API call, and action handlers to manage the response.

```
import {createCustomElement} from '@servicenow/ui-core';
import snabbdom from '@servicenow/ui-renderer-snabbdom';
import {createGraphQLEffect} from '@servicenow/ui-effect-g
```

```
raphql';

const GQL_QUERY = `
query query ($active: Boolean) {
    now {
        mySchema (active: $active) {
            myField {
                value
            }
        }
    }
}`;

const DATA_FETCH_REQUESTED = 'DATA_FETCH_REQUESTED';
const DATA_FETCH_STARTED = 'DATA_FETCH_STARTED';
const DATA_FETCH_SUCCEEDED = 'DATA_FETCH_SUCCEEDED';
const DATA_FETCH_FAILED = 'DATA_FETCH_FAILED';

const dataFetchHandler = createGraphQLEffect(GQL_QUERY, {
    variableList: ['active'],
    startActionType: DATA_FETCH_STARTED,      // gql http request started
    successActionType: DATA_FETCH_SUCCEEDED // gql request succeeded with result
    errorActionType: DATA_FETCH_FAILED       // gql http network request error (actual schema / server errors will succeed with error objects
});

const component = createCustomElement('now-component', {
    renderer: {type: snabbdom},
    view(state) {
        if(state.loading)
            return <div>Loading...</div>;
        if(!state.data)
            return <div>No Data</div>;
        return <div>My value: {state.data.data.now.mySchema.myField.value}</div>;
    },
    actionHandlers: {
        [DATA_FETCH_REQUESTED]: dataFetchHandler,
        [DATA_FETCH_STARTED]: ({updateState}) => updateState({l
```

```
oading: true}),
  [DATA_FETCH_SUCCEEDED]: ({action, updateState}) => updateState({data: action.payload}),
  [DATA_FETCH_FAILED]: () => { /* handle network error */
/ },
}
});
```

GraphQL system properties

Configure GraphQL API framework behavior. For example, you can configure whether to allow introspective queries into your schema.

Navigate to **System Web Services > GraphQL > Properties** to change your GraphQL settings.

Name	Description
glide.graphql.introspection_enabled	<p>Enables introspective queries to discover the supported GraphQL queries and types.</p> <ul style="list-style-type: none">• Type: Boolean• Default: False