Integration Made Easy!

Cloudaction LLC

http://getactionHub.com

Contents

1. Introduction to actionHub

We created actionHub so you could have all the benefits of flexible integration without the hassle, expense, and complexity of custom code. We built it entirely on Salesforce platform so your data never needs to leave the world's most trusted, secure, and reliable cloud. With actionHub, you can stand up a reliable and elegant integration in hours or days instead of weeks or months. It's that simple.

The Challenge

In recent years, we noticed that customers kept coming to us with the same pain points around integration. Despite the different integration options available, they were not able to respond quickly to changing business needs:

- Custom integration solutions are expensive, time-consuming to support, and often lack basic error recovery features. Even simple modifications require a lot of time and money.
- Fancy middleware platforms designed to do everything for everyone are not delivering on the promise to make this easier. They have proprietary scripting languages and so many bells and whistles that specialized training or resources are needed.
- Organizations without big budgets are either out of luck or stuck with a one size fits all integration. Often marketed as "recipes" or things that snap, zip, or pop together, the firms selling them are peddling hundreds of integrations right out of the box. In reality, they are cookie-cutter solutions which lack the flexibility to do the job right. In the real world, businesses are unique and so are their integrations.

The Solution

90% of the time, successful integration is about getting two systems to talk and doing some basic mapping or translation of the data so things line up on both sides. We designed actionHub to make it easy to do that with minimal effort and complexity.

What actionHub Does

actionHub is an integration solution built on the Salesforce platform. It makes it easy to automatically move data between applications in real time. By eliminating redundant manual data entry, actionHub orchestrates data movement so your people will have the right information in the right tool at the right time.

actionHub enables seamless integration for cloud-centric and hybrid enterprise application environments. actionHub has pre-built connectors to quickly integrate leading enterprise applications. You can use our standard REST adapter to integrate with most systems without a special connector. Most importantly, the integrations can be configured for your unique business needs.

actionHub enables you to establish simple integration rules. Each rule defines conditions (when to send data) and actions (what to do with that data). Each action defines a target (where to send) and mappings (what to send). When mapping data which may not be an exact match, actionHub provides tools to perform basic transformation.

Once you define the rules, actionHub takes care of everything else so you can spend your time doing the work you love.

2. Getting Started with actionHub

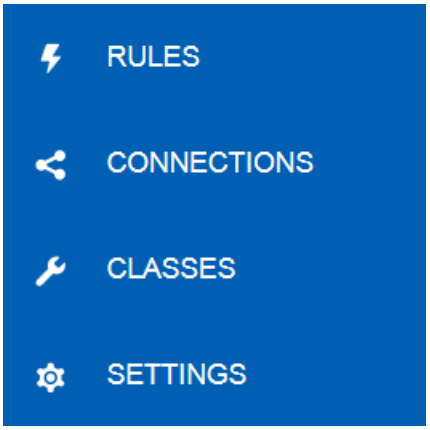
As easy as 1-2-3

Getting actionHub up and running is simple:

1. Create a Connection User with an actionHub license and make it a member of the actionHub Connection Users permission set.
2. Create a Connection (external integration) or assign your Connection User to the Salesforce connection (internal integration).
3. Create Integration Rule(s) and deploy event triggers for internal source objects.

Navigation

Getting around in actionHub is easy. All configuration activity is performed on a few simple pages and they are all available from a simple dropdown menu on the actionHub tab.



The other tabs in actionHub are for reviewing/monitoring integration activity by going directly to the Event Queue, Action Queue, or Logs.

3. Users

Who is a User?

The real users in actionHub are basically service users that represent the specific connections you have configured. Any external system integrated using actionHub will login using the credentials of the user associated with its connection.

As an admin, a consultant, or a developer tasked with the success of an integration project, you will use the actionHub user interface to configure everything. Once you complete the initial setup, you may not open it again until the business requests some enhancement or you are ready for your next integration success.

Most of the users in your Salesforce org will never see or interact with actionHub. It will simply be there, behind the scenes, making them more productive. By ensuring they have the right information at the right time and enabling seamless processes between teams using different apps, actionHub delivers real results without being seen.

Configuring Connection Users

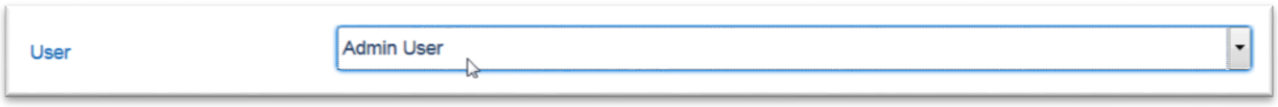
First, identify or create a service user for each connection. Then, simply assign the *actionHub Connection Users* permission set and allocate an actionHub managed package license. That user may now be associated with a connection in actionHub.

While actionHub does not require users to have any specific Salesforce license type, you should ensure that the user has been configured with a license with contractual access to the objects needed for your integration use case.

Assigning Connection Users

If you are configuring an internal integration (between apps/objects within a single org), you will need to assign an actionHub user to the “Salesforce” connection. This will enable you to select that connection as both the source and the target in a rule.

If you are configuring an external integration, you will need to assign your actionHub user to the connection created for the remote system.



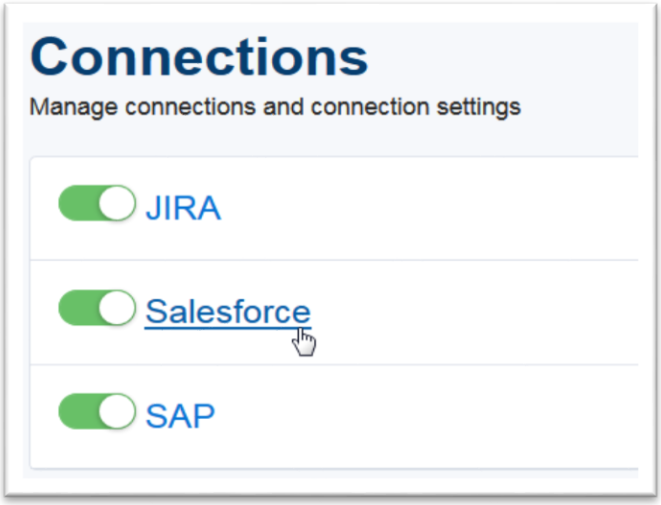
4. Connections

What is a Connection?

Connections represent the endpoints for your integrations. Each rule will have a specific source connection and each action in a rule will have a specific target connection.

When you first install actionHub, it comes with the “Salesforce” connection which provides access to all the objects in your org. You will add other connections as needed for each target system.

You can click on the name of a connection to configure it or use the switch to enable/disable the connection.

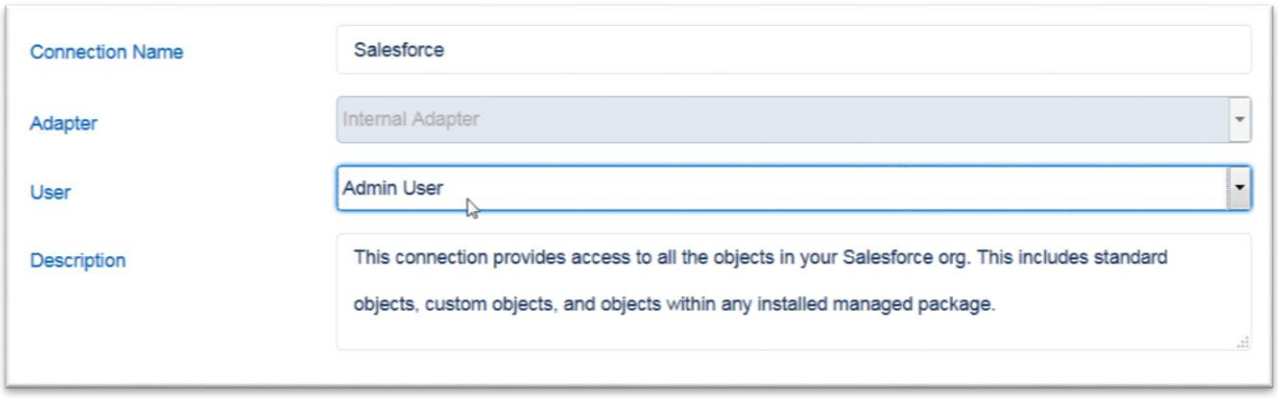


Creating Connections

When configuring any new connection, you will give it a unique name and select an Adapter. When actionHub is first installed, an Internal Adapter and a REST Adapter come pre-configured. These are powerful and can be used to complete many different integration scenarios.

You may also be using a specific adapter designed to make it even easier for you to complete an integration. If needed, you could even build a custom adapter for a homegrown system or special use

The Connection Information tab contains the information common to every adapter:



Each adapter may have different settings it defines. The actionHub user interface displays adapter settings dynamically. For example, the REST Adapter has two additional tabs:

5. Rules

At the heart of actionHub are the integration rules. Each rule defines a Source (where data comes from), Conditions (when to do something) and Actions (what to do with that data).

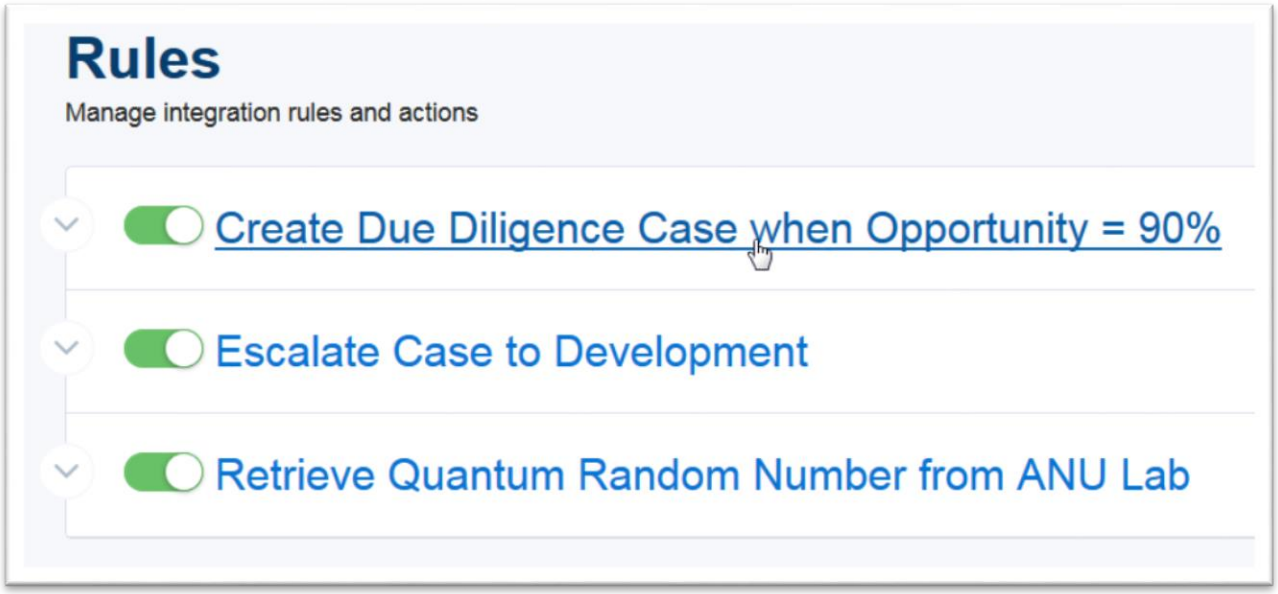
Each Action defines a Target (where to send data), Parameters (settings for the target adapter), Field Mappings (what data to send), and Response Mappings (data to get back from the target).

When mapping data which may not be an exact match, actionHub provides the Expression Builder with Mappings and other tools to perform basic transformation (Functions).

Once you define the rules, actionHub takes care of everything else so you can spend your time doing the work you love.

Rule Explorer

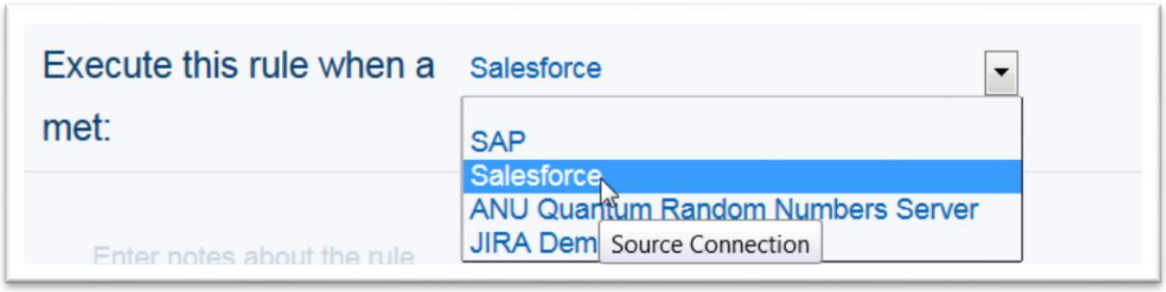
The Rule Explorer is a quick way to see the rules configured in your system. From here you can click to open one in the Rule Editor, use the switch to quickly turn it on/off, or print rule(s) for project documentation or future reference.



Source

Source Connection

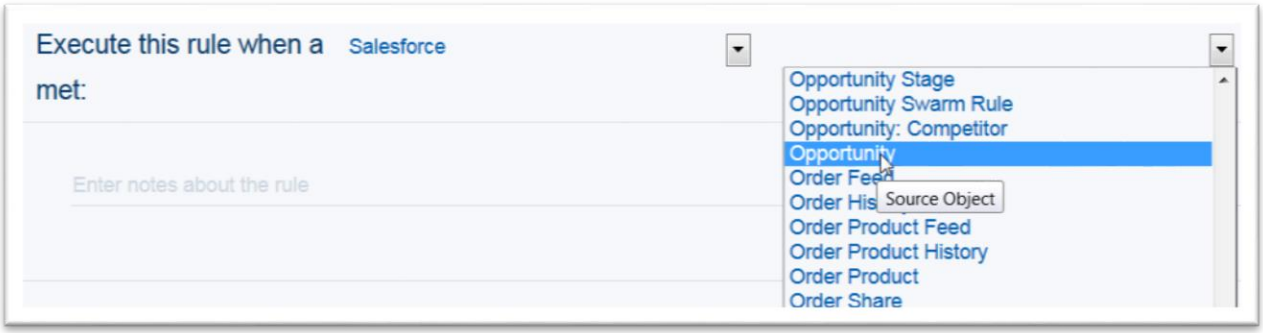
Every rule is associated with a specific Source Connection which is selected using a picklist.



Source Object

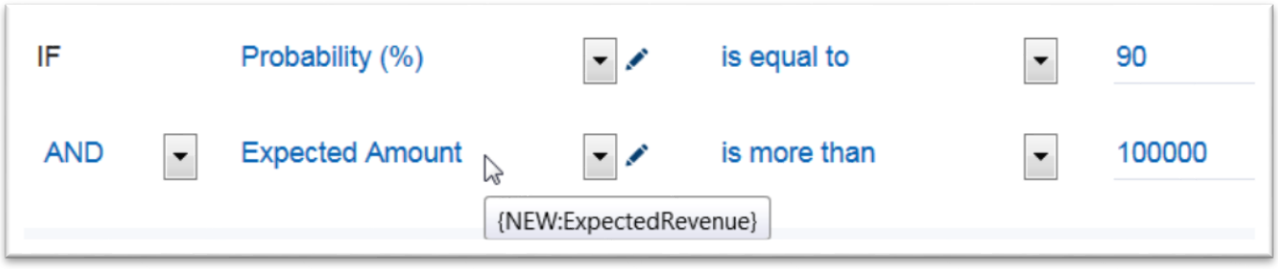
The Source Object represents the place where the initial action begins. For integration events beginning inside the org, this is the object where a record creation or update will trigger a rule. For other systems, the object may also represent an object, database table, or a type of activity.

Most adapters will provide a drop-down list. If one is not provided, you can simply type in the name:



Conditions

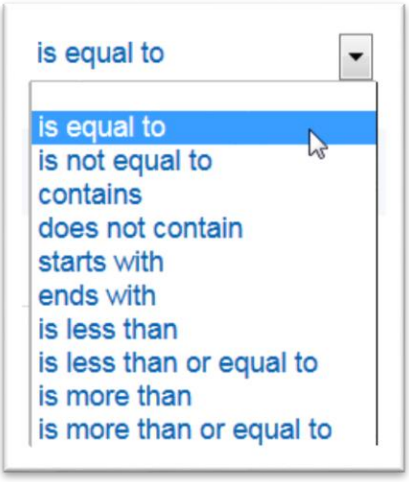
Conditions are used to define when a rule should fire.



For adapters with metadata, the left expression will be a picklist of fields in the source object. If you hover over any selected field, it will display the API name of the field.

The right expression can be populated with a specific value or you can use the Expression Builder to select another field or use mappings/functions for more complex conditions.

The sides are compared using one of the selectable comparison operators.

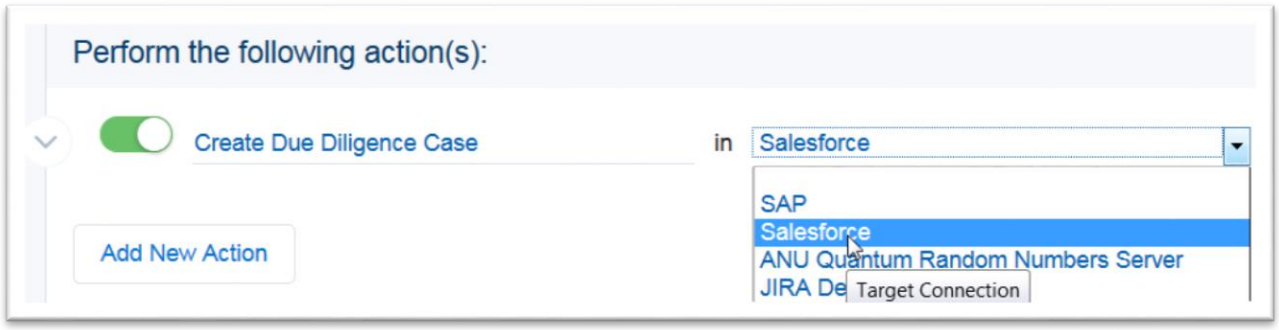


Actions

When a rule meets the specified conditions, actionHub will execute all of the rule actions. Each Action defines a Target (where to send data), Parameters (settings for the target adapter), Field Mappings (what data to send), and Response Mappings (data to get back from the target).

Target Connection

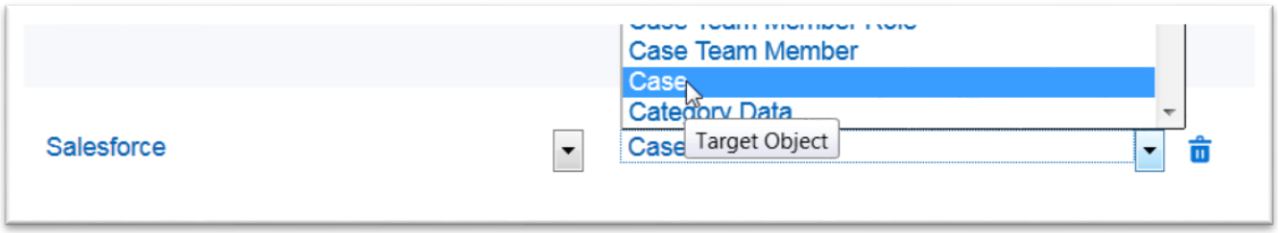
Every action is associated with a specific Target Connection which is selected using a picklist.



NOTE: If you are building an internal integration and you are unable to select Salesforce as your target, you need to go add a Connection User to the Salesforce connection.

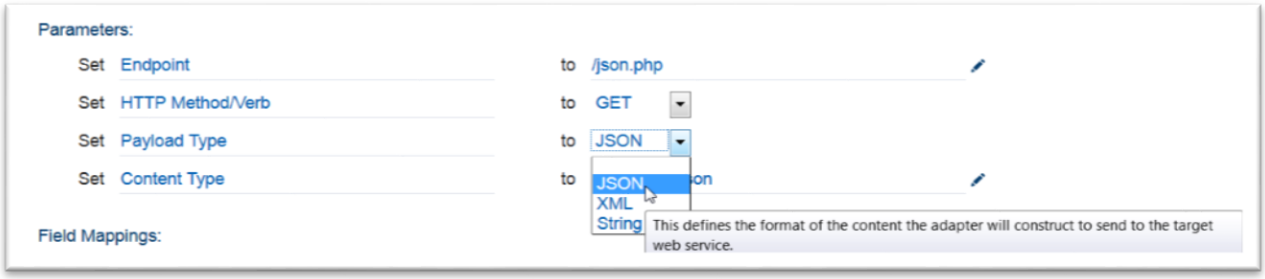
Target Object

For integration events with an internal action, this is the object where a record creation or update will trigger a rule. For other systems, the object may represent an object, database table, or even a type of activity. Most adapters will provide a drop-down list. If one is not provided, you can simply type in the name:



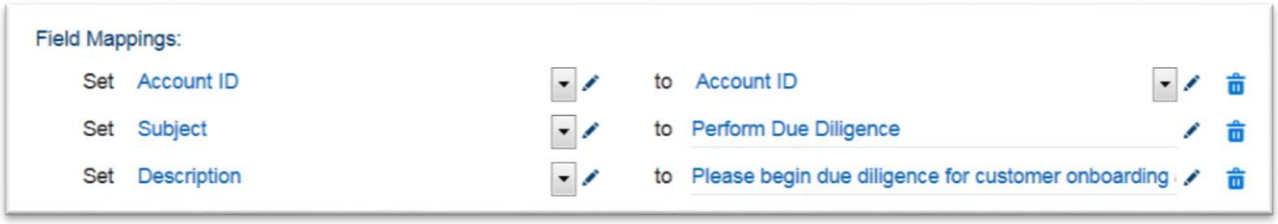
Parameters

Parameters are rule settings which are defined by the adapter to collect needed information needed to handle actions properly. Some adapters (like the internal adapter) have no parameters, while others (like the REST Adapter) may have several. Tooltips and adapter documentation should provide specific guidance on these settings.

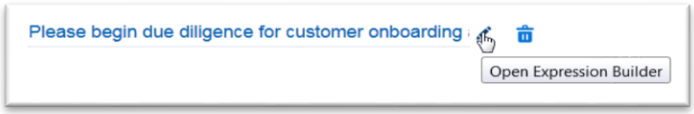


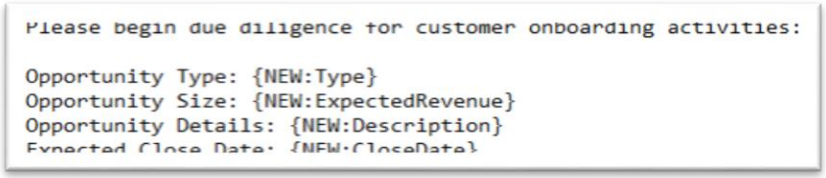
Field Mappings

Field mappings define the data to be applied to each field in the target object. You can select fields for straight-across mappings or enter simple values.



You can also combine values or build complex data using the Expression Builder:



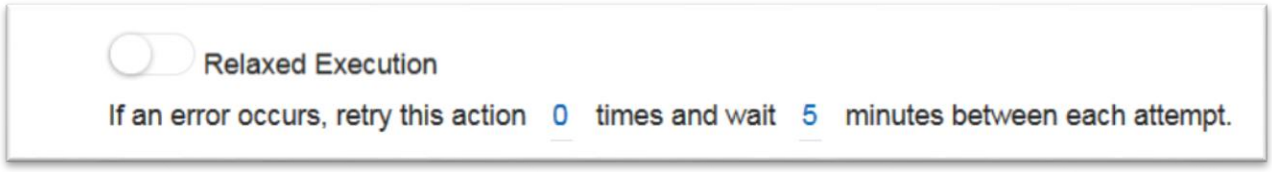


Response Mappings

Response Mappings provide a simple way to update the source record with data from the target record that was created/updated as a result of this action.

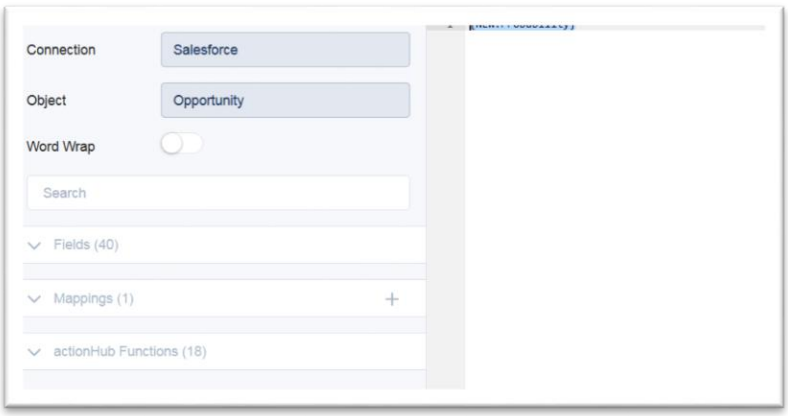
Action Settings

Action Settings allow you to set retry options for each rule. Be sure to consider the capabilities and weaknesses of any target system when enabling retries. Some systems may fail to respond sometimes even if they have created a record. If you retry the action, you could create multiple records!



Relaxed Execution is great for complex actions which are not time-sensitive. It separates the action processing into its own transaction. The Action Sentinel will pick them up for processing.

Expression Builder



The Expression Builder is used to make it easy to perform data transformation for field mappings or build complex expressions for conditions. The left side contains a set of tools and the right side contains the expression you are building.

Fields

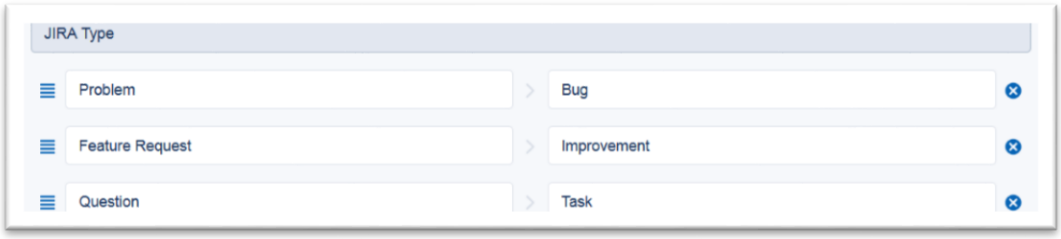
Depending upon your context and the adapter for your connection, there may be a list of fields. If so, clicking on the name of any field will insert a reference to that field into the Expression window. This will appear as {NEW:FieldName}. If you need the old value of a field (before the event), you can click the icon next to the field name and a reference will be inserted with {OLD:FieldName}.

Mappings

Mappings provide a simple way to translate values between systems that use different terminology. When you click on the name of a mapping you have created, a reference will be populated into the Expression window: {MAPPING:"Mapping Name","*InputValue*"}. Just highlight *InputValue* and click on the name of the input field from the field list. The final formula will look something like this: {MAPPING:"Mapping Name","{NEW:FieldName}"}

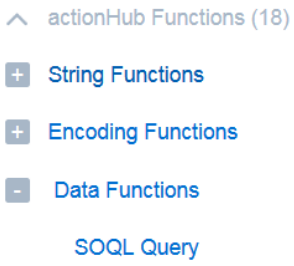
Create New Mapping

From the Expression Builder, just click on the + next to Mappings to add a new Mapping or click the pencil icon to edit an existing mapping. Just give the mapping a name and specify the input value and the value it should become when it is sent to the target system.



Functions

Functions are tools for accessing or transforming data throughout the rule. Tooltips are provided for each function. When you click on the name of any function, a signature for the function will be inserted into the Expression window. Simply highlight placeholder values (e.g. *Source_String*) and click the name of a field and you will have a complete expression like this: {TRIM:"{NEW:FieldName}"}.



Don't be intimidated by more complex functions with multiple arguments (e.g. SOQL Query). The signature and the tooltips will usually tell you everything you need to know!

Trim

Removes leading and trailing whitespace from the "Source String"

Expression: {TRIM:"Source_String"}

Sample: {TRIM:"{NEW:Description}"}

If the Description contained " 123 ", the sample expression would return "123".

Left

Returns the first "Length" number of characters from the "Source String"

Expression: {LEFT:"Source_String","Length"}

Sample: {LEFT:"{NEW:Description}","45"}

If Description contained “This description will fit in the target field after it is shortened”, the sample expression would return “This description will fit in the target field”.

Left Pad

Returns a String with "Padding String" added to the left side of "Source String" until it is "Length" characters long.

Expression: {LPAD:"Source_String","Length","Padding_String"}

Sample: {LPAD:"{NEW:CaseNumber}","10","0"}

If Case # contained "00001234", the sample expression would return a value of "0000001234".

Sample: {LPAD:"{NEW:Status}","8","-"}

If Status contained "PENDING", the sample expression would return a value of "-PENDING".

Right Pad

Returns a String with "Padding String" added to the right side of "Source String" until it is "Length" characters long.

Expression: {RPAD:"Source_String","Length","Padding_String"}

Sample: {RPAD:"{NEW:CaseNumber}","10","x"}

If Case # contained "00001234", the sample expression would return a value of "00001234xx".

Replace

Returns "Source String" where all occurrences of "Search String" have been replaced with "Replacement String".

Expression: {REPLACE:"Source_String","Search_String","Replacement_String"}

Sample: {REPLACE:"{NEW:Description}","Middleware","actionHub"}

If the Description was "Middleware is the best solution for Salesforce integration. Middleware is great!", the expression would return "actionHub is the best solution for Salesforce integration. actionHub is great!".

Replace All

Returns "Source String" where all occurrences of "Search String" have been replaced with "Replacement String". Search string may contain regular expressions. *NOTE: Regular Expressions provide a powerful tool when used with this function. The sample below will convert a linefeed-delimited list to a semi-colon delimited list. For more details about how to use Regular Expressions (RegEx), check out this resource.*

Expression: {REPLACE:"Source_String","Search_String","Replacement_String"}

Sample: {REPLACE:"{NEW:ItemList__c}","\\n",";"}

Item List Value	Expression Result
Item1 Item2 Item3	Item1;Item2;Item3

Right

Returns the last "Length" number of characters from the "Source String"

Expression: {RIGHT:"Source_String","Length"}

Sample: {RIGHT:"{NEW:Phone}","8"}

If Phone contained “(918)555-1212", the sample expression would return “555-1212”.

SubStr

Returns the "Length" number of characters from the "Source String", starting at the position indicated by the "Start Index" (0-based index). *NOTE: A 0-based index means that "0" will return the 1st character, "1" will return the 2nd character, etc.*

Expression: {SUBSTR:"Source_String","Start_Index","Length"}

Sample: {SUBSTR:{NEW:Phone}","1","3"}

If Phone contained “(918)555-1212", the sample expression would return “918”.

SubString

Returns the characters from the "Source String", starting at the position indicated by the "Start Index" (0-based index) and ending at the position indicated by the "End Index" (0-based index). *NOTE: A 0-based index means that "0" will return the 1st character, "1" will return the 2nd character, etc. The IndexOf function may be used to find the specific starting and/or ending position to make this function useful in strings of variable length.*

Expression: {SUBSTRING:"Source_String","Start_Index","End_Index"}

Sample: {SUBSTRING:{NEW:Phone}","5","7"}

If Phone contained “(918)555-1212", the sample expression would return “555”.

IndexOf

Returns the position (0-based index) within "Source String" of the first occurrence of "Search String". *NOTE: A 0-based index means that "0" will return the 1st character, "1" will return the 2nd character, etc. This function is often used with MID or SUBSTRING functions which require one or more Index parameters.*

Expression: {INDEXOF:"Source_String","Search String"}

Sample: {INDEXOF:"{NEW:Phone}","918"}

If Phone contained “(918)555-1212", the sample expression would return “1”.

Mid

Returns the "Number of Characters" from the middle of "Source String", beginning with "Starting Position" (0-based index). *NOTE: A 0-based index means that "0" will return the 1st character, "1" will return the 2nd character, etc.*

Expression: {MID:"Source_String","Starting_Position","Number_Of_Characters"}

Sample: {MID:"{NEW:CaseNumber}","3","5"}

If CaseNumber contained "00001234", the sample expression would return "01234".

Upper Case

Converts all letters in "Source String" to UPPER CASE letters.

Expression: {TOUPPERCASE:"Source_String"}

Sample: {TOUPPERCASE:"{NEW:Status}"}

If Status contained "In Progress", the sample expression would return "IN PROGRESS".

Lower Case

Converts all letters in "Source String" to lower case letters.

Expression: {TOLOWERCASE:"Source_String"}

Sample: {TOLOWERCASE:"{NEW:Status}"}

If Status contained "In Progress", the sample expression would return "in progress".

Reverse

Returns "Source String" with all characters in reverse order.

Expression: {REVERSE:"Source_String"}

Sample: {REVERSE:"{NEW:CaseNumber}"}

If CaseNumber contained "00001234", the sample expression would return "43210000".

Length

Returns count of the characters in the "Source String".

Expression: {LENGTH:"Source_String"}

Sample: {LENGTH:"CaseNumber"}

If CaseNumber contained "00001234", the sample expression would return "8".

URL Encode

Returns a UTF-8 encoded version of "Source String" that is safe to pass in a URL. This method uses the supplied encoding scheme to obtain the bytes for unsafe characters. For more information about the format, see The form-urlencoded Media Type in *Hypertext Markup Language - 2.0*.

Expression: {URLENCODE:"Source_String"}

JSON Encode

Returns an encoded version of "Source String" that is safe to use in a JSON string value.

Expression: {JSONENCODE:"Source_String"}

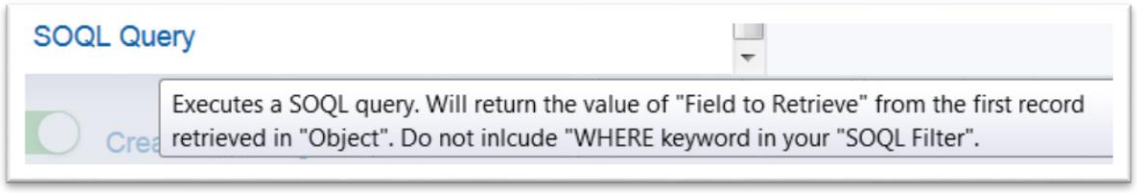
HTML Encode

Returns an encoded version of "Source String" that is safe to use in HTML output.

{HTMLENCODE:"Source_String"}

SOQL Query

Performs a SOQL query to retrieve a value from a record in the Salesforce org. NOTE: The query will always be limited to retrieve a single value from a single record.



Expression:
{SOQL:"Object","Field_to_Retrieve","SOQL_Filter","Order_by_Field","Order_by_Direction"}

Sample: {SOQL:"Case","Description","CaseNumber = '{NEW:ParentId}'","",""}

In the above sample, if the "ParentId" field in the source record was a valid Case Id, the expression would return the Description of the Case.

Sample: {SOQL:"Attachment","Id","ParentId = '{NEW:Id}'","Id","Desc"}

In the above sample, the expression would return the Id of the most recently created Attachment associated with the source record. NOTE: If there were no attachments, the expression would return an empty string!

Deploying Event Triggers for Salesforce Objects

In order for actionHub to begin processing events from any given Salesforce object, you will need to deploy a very simple trigger:

Trigger Template

```
 trigger TriggerName on ObjectName (after insert, after update)
{
        new actionHub.InternalAdapter().acceptEventData(trigger.new, trigger.old,
'Salesforce');
}
```

Example Trigger for Case:

```
trigger actionHub_Case on Case (after insert, after update)
{
        new actionHub.InternalAdapter().acceptEventData(trigger.new, trigger.old,
'Salesforce');
}
```

NOTE: A trigger is only needed for objects which will be the source object in rules and is not needed for target objects used in Actions. This trigger is also not needed for the Attachments object which is handled with a trigger included in the actionHub package.

6. Settings

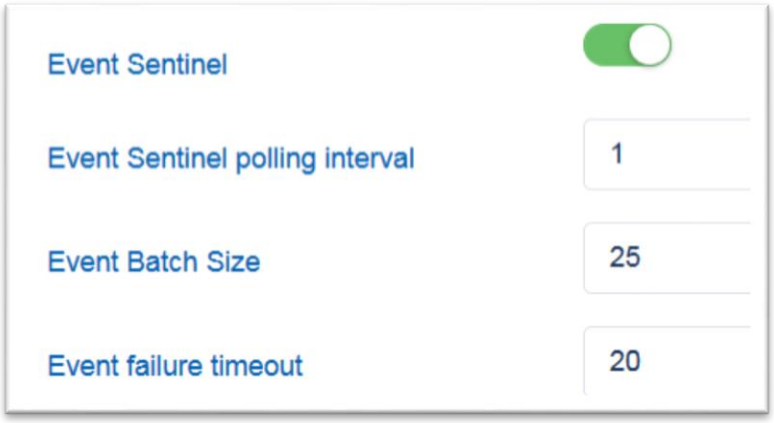
Sentinel Settings

Event Sentinel

When actionHub receives an event from a source connection and there is at least one rule associated with the connection & object, it will create an event queue record.

When data events are performed in asynchronous transactions or in large batches, actionHub places them into the event queue in “relaxed execution” mode. The Event Sentinel will come along and process those events.

When you open Settings, the switch next to Event Sentinel will show you whether it is currently running or not. To enable or disable the sentinel, simply flip the switch and click Save. actionHub will automatically stop/start the sentinel.



Event Sentinel Polling Interval

This is how many minutes the system will wait before rescheduling the Event Sentinel.

Event Batch Size

This determines how many records the Event Sentinel will evaluate in a single transaction. Depending upon the complexity of your rules, this could be hundreds or just a handful. Each record is evaluated against the rules individually. If you need multiple SOQL queries or are running complex functions in conditions, you may need to reduce the batch size. If your rules are fairly simple, you could set this higher.

Event Failure Timeout

This determined how many minutes to wait before considering an event failed. Each action has its own retry interval and failure thresholds and events can only get "stuck" if rule evaluation fails due to a system exception. While very unlikely, the maintenance sentinel will keep things tidy if a custom adapter offers a less-than-graceful exit.

Action Sentinel

When actions are configured for "relaxed execution" or created within an asynchronous transaction, actionHub places them into the action queue in relaxed execution mode. The Action Sentinel will come along and process those actions. It will also take care of retries for any actions which were unsuccessful.

When you open Settings, the switch next to Action Sentinel will show you whether it is currently running or not. To enable or disable the sentinel, simply flip the switch and click Save. actionHub will automatically stop/start the sentinel.

Action Sentinel Polling Interval

This is how many minutes the system will wait before rescheduling the Action Sentinel.

Action Failure Timeout

This determined how many minutes to wait before considering an action failed. If your integration includes a long-running process, be sure to set either this or the Rule's Retry interval to a high enough value to accommodate that process. actionHub will honor the higher of the two values for any given action.

Maintenance Sentinel

The Maintenance sentinel performs routine maintenance to keep the Event Queue, Action Queue, and Logs cleaned up.

When you open Settings, the switch next to Action Sentinel will show you whether it is currently running or not. To enable or disable the sentinel, simply flip the switch and click Save. actionHub will automatically stop/start the sentinel.

Maintenance Sentinel Polling Interval

The polling interval will determine how often it runs.

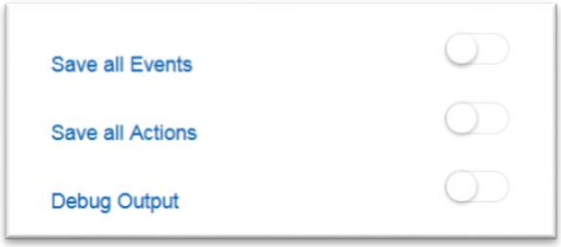
Logging & Retention Settings

Save All Events

When disabled, the system will automatically discard any event queue records if the event does not match conditions for any rules. If Save All Actions is also disabled, the maintenance sentinel will discard events once all actions have been completed. When enabled, the system will keep all event queue records.

Save All Actions

When disabled, the system will automatically discard any action queue records once the actions have been successfully completed. If Save All Events is also disabled, the maintenance sentinel will automatically discard events once all actions have been completed. When enabled, the system will keep all event queue records.



Debug Output

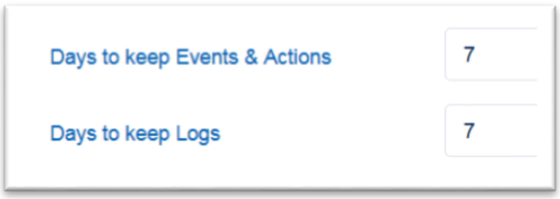
When disabled, the system will maintain minimal logging information (errors only). When enabled, the system will create log entries with a lot of detail which may be useful when initially configuring or troubleshooting your integration.

Days to keep Events & Actions

The maintenance sentinel will automatically discard old events/actions after the number of days specified.

Days to keep Logs

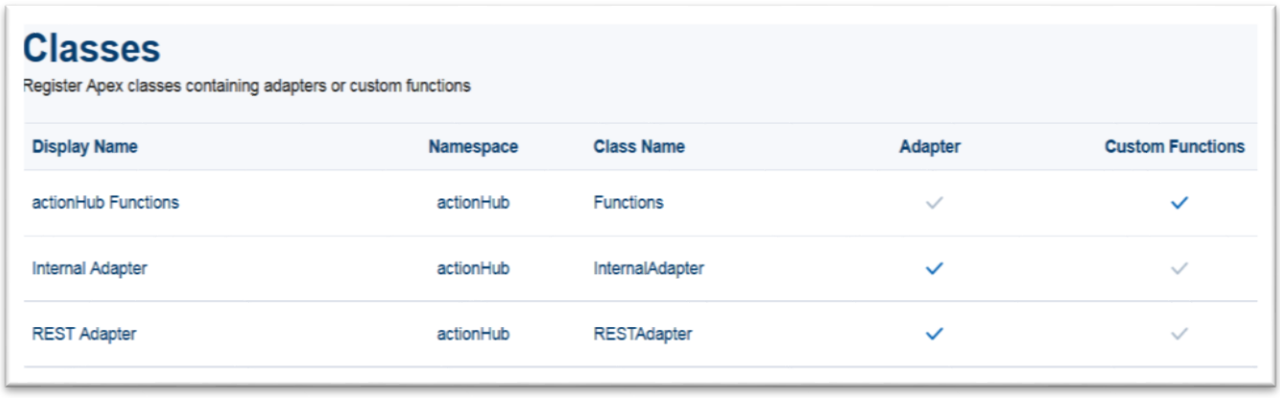
The maintenance sentinel will automatically discard old log entries after the number of days specified.



7. Classes

What are Classes?

A registered class can be used as a custom adapter, a source for custom functions, or both. When you first install actionHub, it comes with a few things already registered.



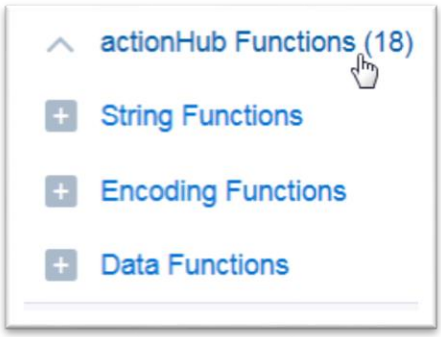
Custom Functions

Custom Functions provide data transformation or other complex logical operations. These functions are accessible through the expression builder and may be used in conditions, parameters, or field mappings.

actionHub Functions

The Functions class includes the most commonly used functions for data manipulation. You will find these in the Expression Builder when you are configuring your rules:

- String Functions: Trim, Left, Left Pad, Right Pad, Replace, Right, SubStr, SubString, Index Of, Mid, Upper Case, Lower Case, Reverse, Length
- Encoding Functions: URL Encode, JSON Encode, HTML Encode
- Data Functions: SOQL Query



Adapters

Adapters are designed to enable integration with a specific type of endpoint. While each endpoint may be different, they all do the same job (send/receive data between actionHub and an endpoint and formatting the data in a way the system will understand).

Each adapter may also provide metadata (lists of objects or fields to make rule configuration easier) and specify its own connection settings or rule parameters. All of those components are dynamically included in the actionHub UI.

Internal Adapter

The Internal Adapter is designed to handle all communication with objects on your Salesforce org. It sends data events to actionHub for processing when records are created/updated in the org and it creates/updates records when it receives an action from actionHub.

REST Adapter

The REST adapter performs the same job, listening for events on an incoming web service and sending actions to the REST web service of a remote endpoint. For more information on how to unlock the power of this flexible adapter, see the next chapter.

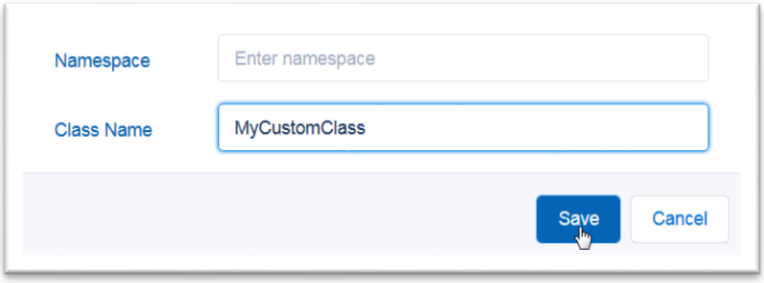
Custom Classes

We designed actionHub to make it easy for you to stand up an integration, but we understand that sometimes things may not be so simple. Rather than load up actionHub with all the bells and whistles you might ever need, we included the tools you need to get the job done 90% of the time.

If you find yourself in one of those 10% situations, don't panic. We know you don't want to ship your data out of the trusted Salesforce platform and off into one of those bloated middleware platforms. There's no need to spend the next 6 months figuring out some new platform when you can just use your existing Salesforce skills and our extensible Class registration.

Custom classes are created in Apex and an SDK with samples is available. To register a class with actionHub, it will need to inherit the actionHub.ICustomFunctions interface and/or the actionHub.ICustomAdapter interface.

Once you have created your class, just go to Classes, click "Add New Class", then enter the Class Name and Namespace (if applicable) and Save. actionHub will automatically validate the class and identify it as an Adapter and/or Functions class.

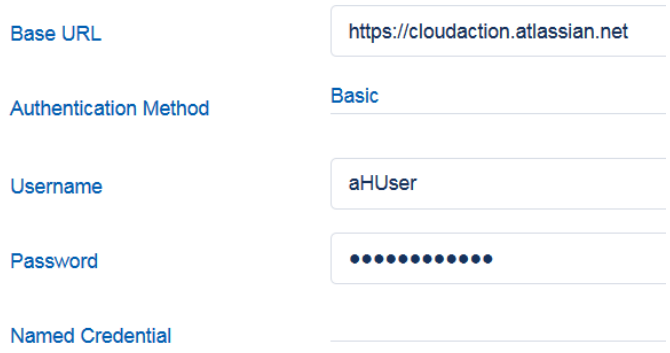


8. REST Adapter

The actionHub REST Adapter is an incredibly powerful and flexible tool designed to make it possible to integrate with the vast majority of modern endpoints without the need for an endpoint-specific adapter to be created!

While each system may require data to be delivered using specified standards (e.g. JSON, XML) and in a very specific format (every system is different), our standard REST adapter provides the ability to easily define the output in a simple way so you don't have to manually assemble the payload (data to be sent).

Connection Settings



Base URL

The Base URL defines the REST web service URL which will be the basis for the complete URL used in any integration operation. Each integration action (defined in the rule action) will have the ability to specify the endpoint, which will be appended to the Base URL to make the complete URL used for the operation.

Be sure to add the Base URL to your list of Remote Sites. Before actionHub can perform a callout to an external site, the site must be registered in the Remote Site Settings page or the call will fail. NOTE: If you are using Named Credentials for authentication and have configured the site's entry for Named Credentials, a Remote Site entry is not needed.

Authentication Method

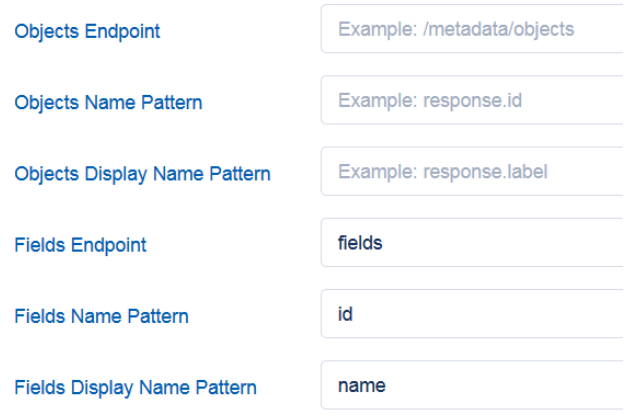
The REST adapter supports the following authentication methods:

- Anonymous (no login)
- Basic Authentication (specify Username/Password)
- Named Credentials (authentication handled by Salesforce platform). See Defining Named Credentials for more information.

Dynamic Metadata

The REST Adapter supports the ability to provide a dynamic list of objects and/or fields from the remote system in some cases (depending upon whether the system can provide that information in a viable format).

To tell actionHub where to retrieve that information, you will specify some simple configuration parameters:



Objects Endpoint

The endpoint is appended to the Connection's Base URL to complete the URL used for the specific integration activity.

If your Base URL is "https://myserver/myapi/" and your Endpoint is Objects Endpoint is "metadata/objects", actionHub will try to retrieve the metadata from "https://myserver/myapi/metadata/objects".

Objects Name Pattern

This pattern specifies where actionHub can find the API Name of the objects in the data returned by the remote system using the Object Endpoint. This pattern is defined using our simple Dot Notation system to define the location of the data in the response from the server. The REST adapter will automatically detect whether the response is in XML or JSON.

For example, if the pattern was "id", actionHub would automatically parse through the response data and retrieve all values associated with that tag (XML) or name/value pair (JSON).

Objects Display Name Pattern

This pattern specifies where actionHub can find the Display Name of the objects in the data returned by the remote system using the Object Endpoint. This pattern is defined using our simple Dot Notation system to define the location of the data in the response from the server. The REST adapter will automatically detect whether the response is in XML or JSON.

For example, if the pattern was "name", actionHub would automatically parse through the response data and retrieve all values associated with that tag (XML) or name/value pair (JSON).

Fields Endpoint

This tells actionHub what to append to the Base URL to retrieve the list of Fields. This can be configured to include a reference to the specified object using a "{OBJECT}" reference. If your Base URL is "https://myserver/myapi/" and your Fields Endpoint is "metadata/{OBJECT}/fields", actionHub will try to retrieve the metadata using the object selected from the object metadata list during configuration. If the object was called "issues", the system would try to retrieve metadata from this URL: "https://myserver/myapi/metadata/issues/fields".

Fields Name Pattern

This pattern specifies where actionHub can find the API Name of the fields in the data returned by the remote system using the Fields Endpoint. This pattern is defined using our simple Dot Notation system to define the location of the data in the response from the server. The REST adapter will automatically detect whether the response is in XML or JSON.

For example, if the pattern was "id", actionHub would automatically parse through the response data and retrieve all values associated with that tag (XML) or name/value pair (JSON).

Fields Display Name Pattern

This pattern specifies where actionHub can find the Display Name of the fields in the data returned by the remote system using the Fields Endpoint. This pattern is defined using our simple Dot Notation system to define the location of the data in the response from the server. The REST adapter will automatically detect whether the response is in XML or JSON.

For example, if the pattern was "name", actionHub would automatically parse through the response data and retrieve all values associated with that tag (XML) or name/value pair (JSON).

Rule Parameters

Endpoint

The endpoint is appended to the Connection's Base URL to complete the URL used for the specific integration activity.

NOTE: For update operations in some systems, you may need to include a field reference in your endpoint. If your Base URL is "https://myserver/myapi/" and your Endpoint is "issue/{NEW:issueNumber} ", actionHub will resolve the URL: "https://myserver/myapi/issue/1234".

HTTP Method/Verb

Specifies the type of operation to the remote server. Options include GET, POST, PUT, PATCH, and DELETE. GET/POST are the most commonly used, but the remote system will define which to use for each operation.

Payload Type

Tells actionHub how to assemble the data for sending to the remote system (XML, JSON, or string). Data will be assembled using our simple Dot Notation system.

Content Type

Specified in the header sent to the remote system. The remote system will generally specify a value to be passed here. If not specified, actionHub will send the default value based upon the selected Payload Type (e.g. application/json, application/xml).

Field Mappings

When configuring field mappings for rules using the REST adapter, the field names you specify will provide the detailed instructions actionHub will need to construct the payload (the data to be sent to the remote system). Field names should be specified using our simple Dot Notation system.

Response Mappings

When configuring response mappings for rules using the REST adapter, the field names you specify will provide the detailed instructions actionHub will need to construct the payload (the data to be sent to the remote system). Field names should be specified using our simple Dot Notation system.

Dot Notation

One of the biggest challenges in integration is that every system has its own requirements for how to format the data and in how it will respond to operations.

To keep things simple, we designed our REST adapter to use a simplified notation system to specify the structure of data in the same way every time. With a few simple settings, the sample methodology will enable you to easily send data to systems with very diverse data structure requirements or even formats/standards.

Whether the system is expecting JSON or XML, our simple system will enable you to map your data quickly and easily.

Basic Concepts

- Use a dot → **.** ← to separate the different structural levels
- Use square brackets → [ ] ← to designate arrays
- Use greater than/less than → <> ←for attributes (XML only)

The following samples will show how Dot Notation is used to construct the payloads. While the samples all show specific values (e.g. "value1") to keep things easy to read, your integrations will most likely be configured with dynamic field values (selected from the list or {NEW:FieldName} format).

The same concept is used in reverse for Response Mappings to define the location to retrieve data in the response from the server.

Example #1 - Simple JSON Output

Field Mappings

Set	request.record1.field1	to	value1
Set	request.record1.field2	to	value2
Set	request.record1.field3	to	value3

Payload Output

```
{
"request":
{
        "record1":
        {
                "field1":"value1"
                "field2":"value2"
                "field3":"value3"
        }
}
```

Example #2 - Simple XML Output

Field Mappings

Set	request.record1.field1	to	value1
Set	request.record1.field2	to	value2
Set	request.record1.field3	to	value3

Payload Output

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
        <record1>
                <field1>value1</field1>
                <field2>value2</field3>
                <field3>value2</field3>
        </record1>
</request>
```

Example #3 - Complex JSON Output (with array)

Field Mappings

Set	request.record[0].field1	to	value1
Set	request.record[0].field2	to	value2
Set	request.record[0].field3	to	value3
Set	request.record[1].field1	to	value4
Set	request.record[1].field2	to	value5
Set	request.record[1].field3	to	value6

Payload Output

```
{
"request":
{
        "record"
        [
                {
                "field1":"value1"
                "field2":"value2"
                "field3":"value3"
                },
                {
                "field1":"value4"
                "field2":"value5"
                "field3":"value6"
                }
}
```

Example #4 - Complex XML Output (with array and attributes)

Field Mappings

Set	request.record[0].<id>	to	1234
Set	request.record[0].field1	to	value1
Set	request.record[0].field2	to	value2
Set	request.record[0].field3	to	value3
Set	request.record[1].<id>	to	5678
Set	request.record[1].field1	to	value4
Set	request.record[1].field2	to	value5
Set	request.record[1].field3	to	value6

Payload Output

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
	<record id="1234">
		<field1>value1</field1>
		<field2>value2</field3>
		<field3>value3</field3>
	</record>
	<record id="5678">
		<field1>value4</field1>
		<field2>value5</field3>
		<field3>value6</field3>
	</record>
</request>
```

9. Monitoring Integrations

The actionHub Event Queue and Action Queue provide a place to see the status of all integration activities. NOTE: the default settings are configured to keep things cleared out in a normally functioning environment. During configuration or troubleshooting, you may want to fine-tune the Logging & Retention Settings!

Events & Event Queue Lifecycle

When actionHub receives an event from a source connection and there is at least one rule associated with the connection & object, it will create an event queue record. Events will go through the following lifecycle:

PREQUEUE

Indicates an event which has just been created but incoming event data is still being saved. It is unlikely you will see a record in this status under normal conditions.

NEW

Indicates an event which has not yet been processed by actionHub.

COMPLETED

Indicates an event which has been successfully processed. This means that the event has been evaluated against rules and any relevant actions have been created. NOTE: Action status is tracked separately in the Action Queue and a COMPLETED event may have pending actions or even failed actions!

FAILED

Indicates an event which failed event processing. This means it could not be evaluated against the rules for that source connection/object and/or the action queue records could not be successfully generated.

Actions & Action Queue

When actionHub rule conditions are met following an event, Action Queue records will be created for each of the actions related to that rule. Actions will go through the following lifecycle:

NEW

Indicates an action which has not yet been processed by actionHub.

IN PROGRESS

Indicates an action which is currently being processed by actionHub. For relaxed execution actions, this status also applies to actions queued for imminent processing by the platform.

COMPLETED

Indicates an action which has been successfully processed. This means that the action has been successfully executed (including processing of any response mappings).

RETRY

Indicates an action which did not process successfully and is pending retry based upon action settings.

FAILED

Indicates an action which failed processing. This will only occur after the action has attempted retries (based on action settings) or if a fatal error occurs (one for which retries will not be successful).

Log

The actionHub log is a great place to collect information when troubleshooting your integrations.

The detailed "Rule Condition Reports" are a great way to see the details of each expression in your conditions and determine whether the overall rule conditions are being evaluated to true (run this rule) or false (don't run this rule). You can also see if the source fields are providing the data you expect.

Many adapters offer a "Callout Report" which provides a great summary of the data from the source event, the data sent to the adapter by actionHub (the output of your field mappings), the data sent to the target system, and the response from the target system. This one report can often tell you everything you need to know to resolve an issue.

NOTE: By default, actionHub will only log errors. When doing initial configuration or troubleshooting, please be sure to check the Logging & Retention Settings!