IntegrationHub Spoke Development

Coding Standards & Best Practices for External Contributors

# Application Scoping & Plugins

* (ServiceNow) scope name should follow the format: sn\_*name*\_spoke e.g. sn\_ad\_spoke
* Create only one Scoped Application for a Spoke – the spoke should be responsible for doing a specific integration. For Ex - Box Spoke should be responsible for doing a native integration with Box in terms of managing files and folders.

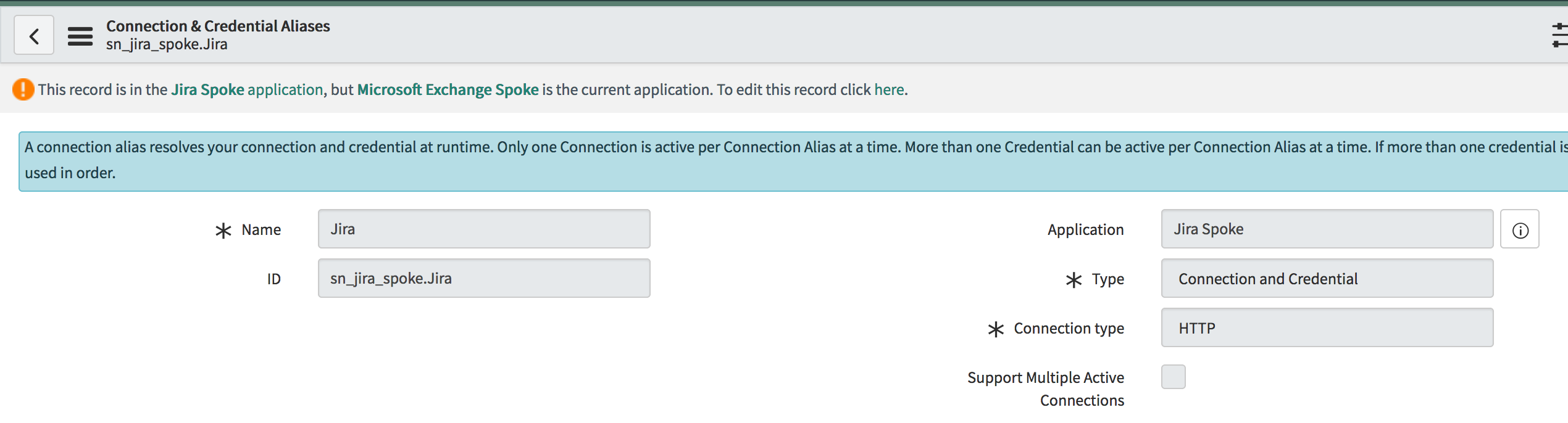
# Authentication and The Connection & Credential Alias

## Default Alias Set up

Every Spoke requires a default Connection and Credential alias. The following are the guidelines:

1. Alias should be created in the scope of the Spoke
2. The name of the alias should be the name of the Spoke. Do not include “spoke”.
3. Add a Connection Attribute for API Version on Connection & Credential Alias with a default value. The preferred label for this attribute is Version.
4. The Spokes which use OAuth setup should ship a default Application Registry
5. Application Registry name should match the spoke name

Eg: Jira Connection and Credential Alias



## Inbound Webhook Authentication / Callbacks

In case of Bidirectional communication between a third-party system and ServiceNow it’s recommended to support three design patterns when possible:

1. Webhooks
   1. Webhooks are implemented as Scripted Rest API protected by an authentication mechanism – either Basic or Custom Token Based.
2. Polling
   1. Using a scheduled flow, check remote system for updates (this is not preferred compared to WebHooks, but some customers do not have access to setup a callback from the the remote system
3. Email
   1. Inbound Email Trigger/Flow
      1. Most remote systems support sending out an email upon an event. Provide example instructions for remote system to setup outbound email (when appropriate), with matching inbound email flow
      2. Inbound email flow can either: receive the lookup ID of the object of interest, then do an outbound call to fetch the details. Or, parse the email for data of interest (less preferred as this relies on perfect parsing of that email).

Every spoke should support all three patterns (as appropriate), giving the customers the choice based on their environment, on which pattern fits best.

The preferred pattern is to expose webhooks from ServiceNow. Webhooks are implemented as Scripted Rest API protected by an authentication mechanism – either Basic or Custom Token Based.

For Ex: - When an Issue is created in Jira or Updated in Jira if you want to invoke a flow at ServiceNow.

Please follow the below steps.

* Create a Webhook in ServiceNow using Scripted Rest API
* Protect the API with an Authentication Mechanism.
* Register the Webhook as a listener in the Targeted System.
* Invoke a Flow when the listener is fired. The Business logic of invoking the flow can be within the listener or outside it.

# Guidelines for creating new Actions

1. Actions should be created in the spokes that are relevant. Don’t duplicate the same action in multiple spokes. E.g. If you have an Azure AD spoke that does user management for all Azure and Office 365 services, when you create a spoke for MS Teams, don’t duplicate user management in Teams. Ensure the Azure AD spoke has what you need. A solution may require support of multiple spokes.
2. By default, set scope access to “Accessible from all scopes” in actions. The scope should only be reduced if a spoke is proprietary to a given application. E.g. Benchmarks may have a spoke that is only ever used by the Benchmark app.
3. Flow Annotations are generally used by a Flow Designer to provide context when authoring a flow. Annotations on an action are optional. Only use annotations if the flow step needs extra description e.g. **do not** have annotation that repeats what the action name or subflow e.g. My Action is Create User, an Annotation that says, “This action will create a user” has negative value and creates noise on the flow.
4. Actions should have a category unless there are only a few actions in the entire spoke
5. Actions must be published state.
6. Every Action **must** have unique name in its scope.

## Action Naming Conventions

1. Action name should be “human readable”, i.e. not a system/function name. Assume the user consuming the action is not highly technical or aware of the APIs.
   1. E.g. “Start Virtual Machine”
2. Words in action names should be separated by a space, every word should begin with an uppercase character.
   1. Correct: “*Upload File*”.
   2. Incorrect: “*UploadFile*” or “*Upload file*”.
3. Action Inputs and Outputs should be “human readable”, i.e. not an internal name with special characters. Assume the user consuming the action is not highly technical or aware of the APIs.
   1. E.g. “Incident List” not “incident\_list”
4. Use 'Look Up' as the prefix if the purpose of the action is to search for object(s), Use ‘Get’ as the prefix if the purpose of the action is to get details of an object given its unique ID.

a. Correct: ‘Get Issue Details By ID’, ‘Look Up Records By JQL’

b. Incorrect: ‘Lookup Issue By ID’,’Get Records By JQL’

## Action Category Guidelines

* Each Action should ideally be associated with a Category unless there are only a 1-2 actions in a spoke.

Ex: For AD spoke any actions relating to users should be categorized under “User Management”.

* It’s ok to keep an action under the default category if it doesn’t belong to any category.

## Step Design Guidelines

### Preprocessing & Error Handler Scripting Steps

* Pre-processing and error handling should be done in separate script includes which need to be called from Action Designer script steps.
* Code repetition should be carefully eliminated (including ‘if’ statement checks) and modularize common functions.
* Use gs.error to log detailed error in case of failures.
* Before every integration step, a best practice is to do a preprocessing step and check for empty values in run time and throw errors.
* Avoid reading contents of attachment in script step to send it in REST step.
* Errors generated from Integration step like REST, PowerShell or SOAP should not be thrown as errors. Instead they should be added to output variable “ErrorMessage”
* Error Handling step must validate all the outputs of REST/POWERSHELL steps for nulls before using them in further processing.
* Provide a default error message which will be used as fall back.
* Remove null inputs from request body --- please remove inputs which are not defined from the request body. There are few 3rd party systems that supported

### REST Steps

* Use Binary type and Multipart type of REST step to transfer attachments to integration system
* Use the Version variable declared in Connection Attributes
* The best practice is to only have a single integration call in an action. Do not include more than a single integration call in an action. An action should only have a single integration step. If there is a desire to have more than one, it should be discussed.

Note: Resource Path in the Rest Step is encoded.

## Error Handling Best practices

* Decouple platform error messages from API error messages - This will reduce cohesion and will enable flow designers to write logic based on platform error messages.
* Make error messages understandable to the Process Analyst e.g. if the error returned from active directory is "**The RPC server is unavailable**", it can be swapped with a more user-friendly error message as "**Invalid hostname or host unreachable**"
* Error messages should be short and self-explanatory.
* Incorporate all error messages that the API returns in the Action description.
* Handling Errors by creating Error Handlers:

Create Error Handlers and abstract the error handling logic to a specific spoke.

The flow designer should throw exceptions explicitly in case of any errors in the pre-processing Step and the framework takes care of handling such errors,

For e.g. if there is step that requires 2 steps (like REST followed by lookup record) , it makes no sense to continue with the lookup record which is dependent on REST step output.

In that case it’s better to prematurely throw an exception:

If(inputs.status\_code != '200'){

*//Throw an Error*

new GenericErrorHandler().throwException(inputs.status\_code,inputs.response);

**Function definition for throwException():**

throwException:**function**(status\_code,message){

var errorMsg = “Some action failed **with** status code :”\_status\_code;

**throw** **new** Error(errorMsg);}

If there are any errors at the API end or at the service with which the flow is interacting with, the error message appears as a pill and it is then the responsibility of the flow designer to employ flow logic to take a decision based on the error message.

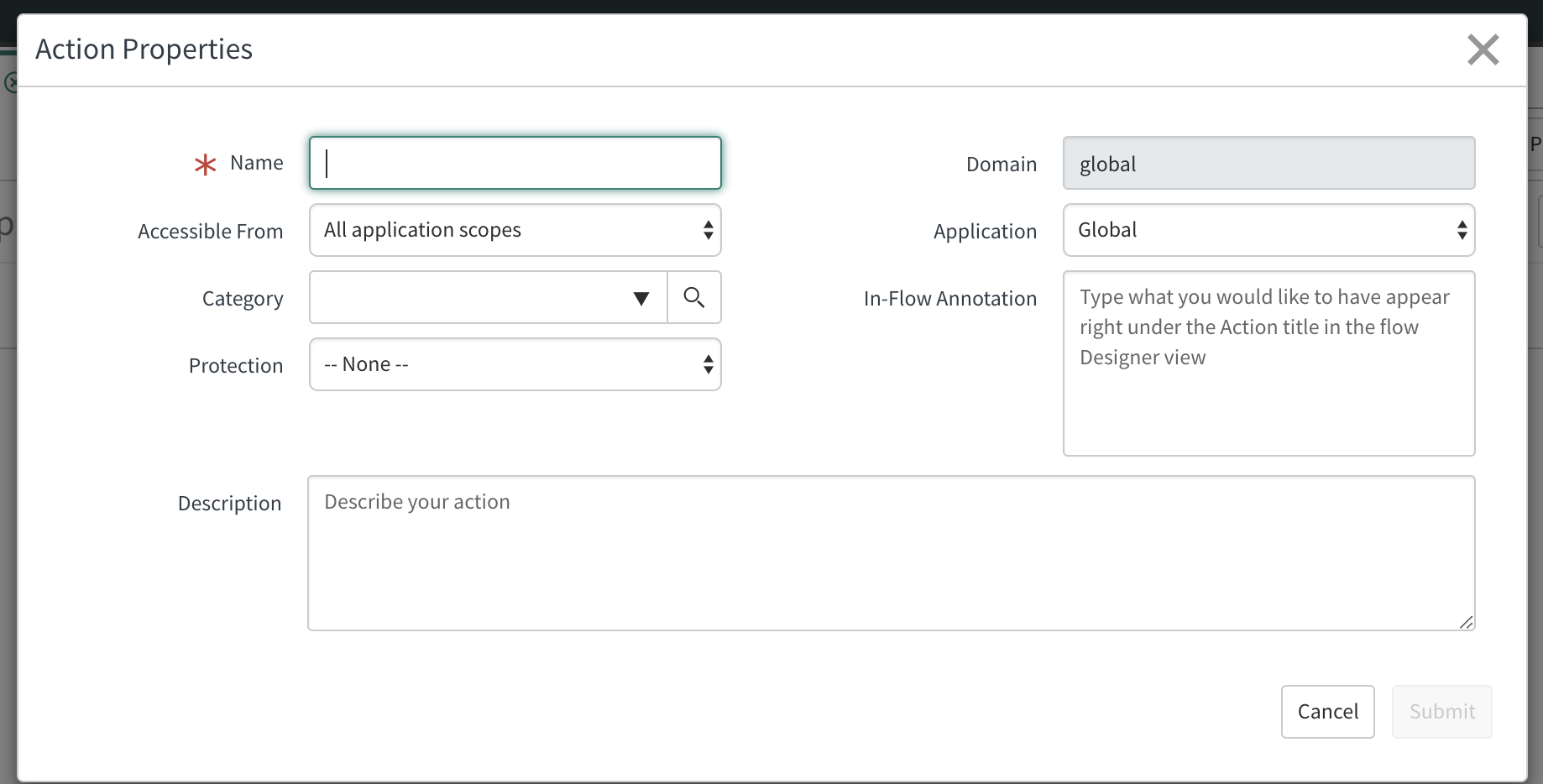
For e.g. If an error occurs which connecting to Azure AD, based on the error message create a task for the fulfiller to work on.

## Script includes Guidelines

* Should not be client callable.
* In case, if script includes need public access, across scope, please use scripted rest api and call script includes from it.
* Always protect REST API with roles.
* Create separate script includes for Utililties and Error Handling
* Script includes should avoid need cross scope access; some use case exceptions are possible

## Creating your First Action:

Navigate to flow designer and click the New button on the top right and select action.



Give the necessary information such as the Action name, Category... etc. Select your scoped app from the application choice list if it’s not the default one. Click Submit.

## Action Steps:

Though all the steps here are not mandatory, A typical action generally has these steps.

* Inputs
* Input Pre-Processing and Validation
* Steps depending on your action – You can use one or more steps depending on your action objective. Ex: REST,Script
* Error/Response Handling
* Outputs

#### Inputs:

Specify the inputs needed for your action here. Each Input has different properties based on its type. Give the Name and type of each field.

#### Input Pre-Processing and Validation:

You can use this step to validate inputs such as If any input contains only spaces and no valid characters. You can also trim the inputs if it’s strictly required by the API.

#### Error/Response Handling:

You can use this step for handling the response from the previous steps (REST), you can also handle different errors and populate the error messages based on the status code and error message.

#### Outputs:

Use this step to specify the final outputs of the action, irrespective of the action, Status as a Choice List and Error Message as a String should always be given as output.

After you have finished making the action, go to properties and make it read only. Give the description to your action in HTML format.

Save and publish your action, based on the version you are using, you may get one or more files generated in your customer update folder for each action.