Concepts and Background -- Create API Key

I have created 4 examples that show how to generate an API Key in SystemLink, 2 by referencing existing Policies and 2 by passing specific Statements, 2 as python scripts you can run in Visual Studio Code or DOS and 2 as JupyterHub notebooks you can run right there in SystemLink.

**Note that you need SuperUser permissions to create ApiKeys**

From SystemLink’s perspective, an ApiKey is just another user

Put differently, Each ApiKey must have a unique name

In the "Access Control" page in the SystemLink GUI, you see Workspaces, Roles, and Users displayed as parallel options, though there is an implicit hierarchy there. The objects in the API that hold that information are more numerous and differently named-- here's a summary attempting to provide clarity:

**GUI: Workspace\User+Roles\Privileges**

Each "Workspace" is a subset of items (Files, Tags, Systems...) that can be acted upon, with an understandable name ("Default")

Each "User" is a log-in account (or logical group of accounts) that is added with one or more Roles to one or more Workspaces

Each "Role" is a collection of atomic Privileges with an understandable name ("Systems Maintainer")

Each "Privilege" is an atomic action that a user assigned that "Role" can perform inside any "Workspace" assigned to that user ("Upload files")

**API: key\user+policies\statements\actions+workspace+resources+description**

Each "api-key" describes a collection of policies, associated to a user, with an understandable name ("Super User Access")

Each "policy" describes a collection of statements that builds a complete set of permissions, with an understandable name ("Super User")

Each "statement" describes a collection of actions with associated workspace, resource list, and understandable description ("File Maintainer")

Each "action" describes an atomic privilege like you see in the Role permissions list in the GUI (file:Query, file:Download, file:Upload, file:Update)

Only Session Keys expire automatically. A regular ApiKey must be deleted or disabled after it is created, if you want those permissions to end.

Only Session Keys require whitelisted authentication. A regular ApiKey can be created with normal web service requests.

The ApiKey definition will always have an associated "userId" property.

The Session Key requires at least "userId" or "orgId" property and will accept both.

The “Super User” Policy is the only Policy with "builtIn" = True.

You can look up the statements and actions of each Role by running the “GET /policy-templates” route with "builtIn" = True.

Find the policyTemplateId corresponding to the Role (from SLS GUI) that you want to use, then search for the policy that has that policyTemplateId in its “templateId” field.  Here’s the “Data Maintainer” policy template on my server:

**{**

**"builtIn": true,**

**"created": "2021-08-24T17:31:09.862Z",**

**"deleted": false,**

**"id": "a256bd29-12ec-4cb0-b59e-19c03c47f3c6",**

**"name": "Data Maintainer",**

**"properties": {},**

**"statements": [**

… and here’s the policy on my server that uses that “Data Maintainer” policy template to define its permission set:

**{**

**"builtIn": false,**

**"created": "2022-04-02T23:50:51.316Z",**

**"deleted": false,**

**"id": "e0dd67ef-81de-43be-9da0-485968587491",**

**"name": "a256bd29-12ec-4cb0-b59e-19c03c47f3c6\_48662d57-e2be-41d7-8865-af5100cd8c6b",**

**"properties": {},**

**"templateId": "a256bd29-12ec-4cb0-b59e-19c03c47f3c6",**

**"type": "role",**

**"updated": "2022-04-02T23:50:51.316Z",**

**"userId": "1e774bea-f42a-4eff-9776-e19401a294fd",**

**"workspace": "48662d57-e2be-41d7-8865-af5100cd8c6b"**

**},**

When I use that policyId ("e0dd67ef-81de-43be-9da0-485968587491") in the definition of my api\_key\_json input, it works on my end:

api\_key\_json = {

  "name": "Data Maintainer Access",

  "policyIds": ["e0dd67ef-81de-43be-9da0-485968587491"],

  "properties": {"ApiKeyType": "programmatic"}

}