# SaveScript

## Description

The **SaveScript** API allows the Account Owner to create or update a *Script*.

> Apstrata provides a core scripting engine using, in a native way, the ubiquitous and powerful JavaScript language. *Scripts* are executed on the server, and they have full access to the apstrata code API. Moreover, they provide the developer with the ability to add more complex validation rules and business logic while interacting with the data. *Scripts* also provide a way to the developer to orchestrate multiple API calls, passing data from one call to another. Each *Script* has its own global scope which gets destroyed at the end of that *Script* execution.
>
> Each *Script* has its own set of read, write, and execute ACLs identifying the Users and/or Groups who can read, write, and execute that script respectively.
>
> A *Script* can be run synchronously by calling the RunScript API or asynchronously by scheduling it to run at a certain date and time. Each *Script* execution time is limited to 30 seconds for safety and security reasons. Hence, *Scripts* operating on large data sets will need to run in asynchronous mode, where a batch of the data is handled every time the script is fired. In this case, the *Script* keeps rescheduling itself until all the data set has been handled.
>
> A developer may choose to generate debug level info from any *Script* and have those log traces downloaded as a file using the GetScriptLogs API.
>
> A developer is also given the choice to make his set of *Scripts* for a particular application mode modular by defining *Script* libraries that can be included in other *Scripts*. This is very useful for building different classes of script libraries that can be integrated in any application, such as social media script libraries, persistence script libraries, identity script libraries, messaging script libraries, etc.
>
> The scripts that are created by the applications' owners are private and can only be accessible under that application. However, Apstrata provides a set of scripts that can be shared among all the applications; this feature grants the application owner the privileges to view those scripts and run them, to import them into their own script libraries and then apply desired updates and modifications on that private copies.
>
> For more specific details on scripts, please refer to the Server Scripting Reference section.

- SaveScript API could be called to import a system script; by passing its name and its content in parameters (system script's name can be gotten from the ListScripts response, and its content can be gotten from the GetScript response). And note that whenever the application owner attempts to update a script that is found in both application and system libraries, only the private script will be updated, the system script will never be affected.

## Script template

```
<script>
        <scriptACL>
                <execute>nobody</execute>
                <read>nobody</read>
                <write>nobody</write>
        </scriptACL>
        <code>
                <![CDATA[
                        //Your code goes here
                ]]>
        </code>
</script>
```

- The execute ACL tag above represents the list of users, devices, groups or identifiers that can execute the saved script. Refer to Data Security page to learn more about ACLs.

## Specific Request Parameters

(Refer to Common Request Parameters)

| Name | Description | Required | Default | Possible Values |
|---|---|---|---|---|

|  |  |  |  |  |
|---|---|---|---|---|
| apsdb.scriptName | Represents the full script name that can consists of up to 5 folders and the script file name separated by the character '/'.<br>*e.g.*, application/provisioning/user/registration<br>In the example above the **registration** script is saved under the three folders **application** > **provisioning** > **user** | Yes |  | • Must begin with an alphabetic character.<br>• Must end with an alphanumeric character or underscore (_).<br>• Can contain alphanumeric characters, underscores (_) and periods (.).<br>• Cannot have two or more consecutive periods.<br>The full script name including any folders must be minimum 3 characters and maximum 128 characters long; it should respect the following regular expression ^([A-Za-z][A-Za-z0-9_])( ([A-Za-z0-9_])+(\Q.\E) )*([A-Za-z0-9_]+) |
| apsdb.newScriptName | The new name of the script (in case of an update). | No |  |  |
| apsdb.update | Specifies whether an update is requested or not. | No | false | true<br>false |
| apsdb.script | The XML definition of the script containing the ACLs and the script code. | No |  |  |

## Specific Logical Errors

(Refer to Common Logical Error Codes)

| Error | Message | Status Code |
|---|---|---|
| SCRIPT_NAME_REQUIRED |  | 400 |
| INVALID_SCRIPT_NAME | Invalid script name [scriptName] | 400 |
| PERMISSION_DENIED |  | 403 |
| SCRIPT_ALREADY_EXISTS |  | 400 |
| INVALID_SCRIPT |  | 400 |
| PARAMETER_REQUIRED |  | 400 |

## Examples

Sample Request

```
Request URL: http://sandbox.apstrata.com/apsdb/rest/[authenticationkey]/SaveScript?apsws.time=[timestamp]
&apsws.authSig=[signature]
```

POST parameters:

```
apsdb.scriptName=[script_name]
apsdb.script= [script_content]
```

Sample XML Response

Success XML:

```
<response xmlns="http://www.apstrata.com/services/schemas/apstrata_database_response.xsd">
    <metadata>
        <requestId>xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx</requestId>
        <status>success</status>
    </metadata>
        <result>
                <localSHA>eb224e0951e68d0a2a227db556041c655c0e17ed</localSHA>
                <metadataLocalSHA>cebbbe640e8f1510069b2fc77494db6ef7b11a36</metadataLocalSHA>
        </result>
</response>
```

Failure XML:

```
<response xmlns="http://www.apstrata.com/services/schemas/apstrata_database_response.xsd">
    <metadata
        <requestId>xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx</requestId>
        <status>failure</status>
        <errorCode>[errorCode]</errorCode>
        <errorDetail>[failMsg]</errorDetail>
    </metadata>
</response>
```

Sample JSON Response

```
{
        "response": {
                "metadata": {
                        "requestId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
                        "status": "success"
                },
                "result":{
                        "localSHA":"eb224e0951e68d0a2a227db556041c655c0e17ed",
                        "metadataLocalSHA":"cebbbe640e8f1510069b2fc77494db6ef7b11a36"
                }
        }
}
```