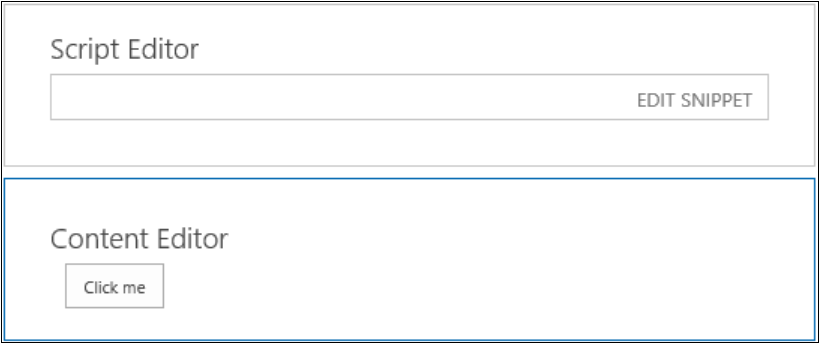
**REST API to Create List in SharePoint**

* REST-API is based on [**OData (Open Data) Protocol**](https://www.codesharepoint.com/interview-question/what-is-odata?) and hence it doesn't require any dll or JS library to run the commands on the SharePoint objects.
* REST-API calls require proper **end-point URL** and proper **Header** implementation in your REST call.

**How to use SharePoint REST API?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* You can also use this method in your **SPFx Component**.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

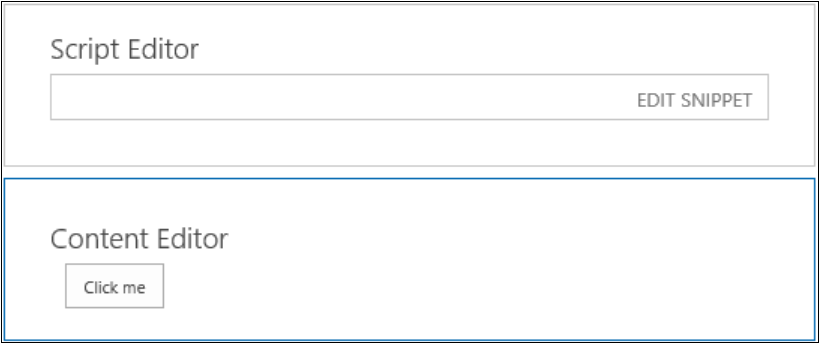
* Once you are done with setting your page, click on the button and your method should get executed.
* function CreateList() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/",
* type: "POST",
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* Title: "List Name",
* Description: "New List Description",
* BaseTemplate: 100
* }),
* headers:
* {
* "Accept": "application/json;odata=verbose",
* "Content-Type": "application/json;odata=verbose",
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Break Role Inheritance On List in SharePoint using REST API**

The example in this topic show how to use REST API to Break Role Inheritance On List in SharePoint

* REST-API is based on [**OData (Open Data) Protocol**](https://www.codesharepoint.com/interview-question/what-is-odata?) and hence it doesn't require any dll or JS library to run the commands on the SharePoint objects.
* REST-API calls require proper **end-point URL** and proper **Header** implementation in your REST call.

**How to use SharePoint REST API?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* You can also use this method in your **SPFx Component**.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

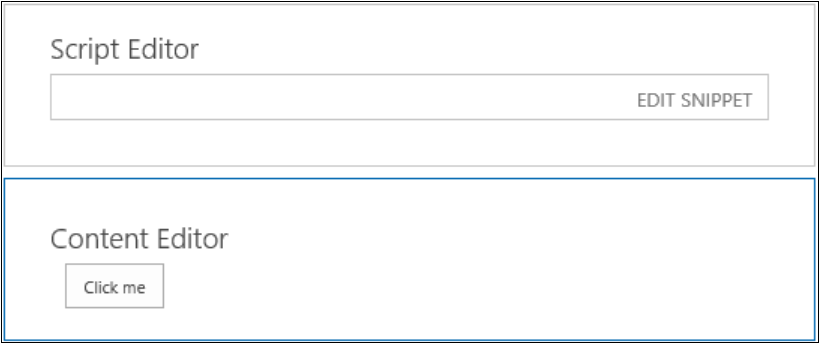
* Once you are done with setting your page, click on the button and your method should get executed.
* function BreakRoleInheritanceOnList()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* // This method takes two parameteres.
* // (1) copyRoleAssignments(Boolean) - Specifies whether to copy the role assignments from the parent or not.
* // If the value is false, the collection of role assignments must contain only 1 role assignment containing the current user after the operation
* // (2) clearSubscopes(Boolean) - clearsubscopes parameter is true, the role assignments for all child objects will be cleared and those objects will inherit role assignments from the current object after this call.
* // If the clearsubscopes parameter is false, the role assignments for all child objects which do not inherit role assignments from their parent must remain unchanged.
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web/lists/getByTitle('List%20Name')/BreakRoleInheritance(copyRoleAssignments=true, clearSubscopes=true)",
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "PATCH",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Delete List in SharePoint using REST API**

The example in this topic show how to use REST API to Delete List in SharePoint

* REST-API is based on [**OData (Open Data) Protocol**](https://www.codesharepoint.com/interview-question/what-is-odata?) and hence it doesn't require any dll or JS library to run the commands on the SharePoint objects.
* REST-API calls require proper **end-point URL** and proper **Header** implementation in your REST call.

**How to use SharePoint REST API?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* You can also use this method in your **SPFx Component**.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

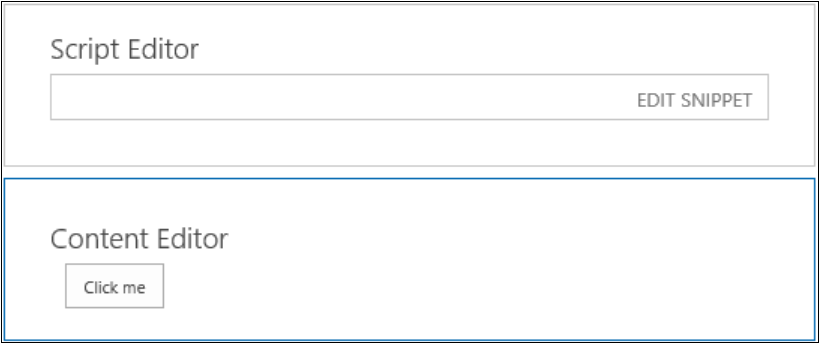
* Once you are done with setting your page, click on the button and your method should get executed.
* function DeleteList() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this will other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('List Name')",
* type: "DELETE",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Remove Groups Permissions From List in SharePoint using REST API**

The example in this topic show how to use REST API to Remove Groups Permissions From List in SharePoint

* REST-API is based on [**OData (Open Data) Protocol**](https://www.codesharepoint.com/interview-question/what-is-odata?) and hence it doesn't require any dll or JS library to run the commands on the SharePoint objects.
* REST-API calls require proper **end-point URL** and proper **Header** implementation in your REST call.

**How to use SharePoint REST API?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* You can also use this method in your **SPFx Component**.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

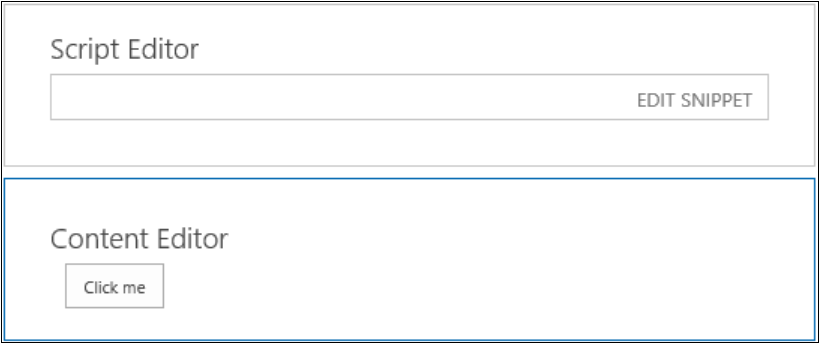
* Once you are done with setting your page, click on the button and your method should get executed.
* function RemoveGroupPermissionFromList()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* // This method will work only if the role inheritance is broken (list has unique role assignments) on the list
* // Provide Login Name of the group that you want to remove
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('List%20Name')/RoleAssignments/groups/RemoveByLoginName('Designers')",
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Set List description in SharePoint using REST API**

The example in this topic show how to use REST API to Set List description in SharePoint

* REST-API is based on [**OData (Open Data) Protocol**](https://www.codesharepoint.com/interview-question/what-is-odata?) and hence it doesn't require any dll or JS library to run the commands on the SharePoint objects.
* REST-API calls require proper **end-point URL** and proper **Header** implementation in your REST call.

**How to use SharePoint REST API?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* You can also use this method in your **SPFx Component**.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

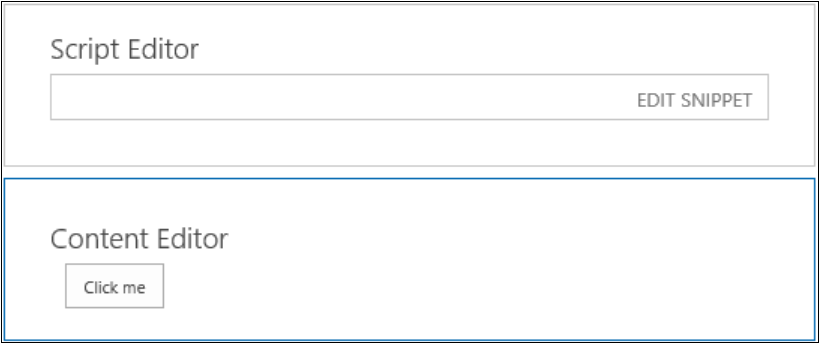
<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

* Once you are done with setting your page, click on the button and your method should get executed.
* function SetListDescription() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('Documents')",
* // The HTTP request method: GET for read operations and POST for write operations.
* // POST requests can perform update or delete operations by specifying a DELETE, MERGE, or PUT verb in the X-HTTP-Method header.
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "MERGE",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* Description: "New List Description"
* }),
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });}

**The example in this topic show how to use REST API to Set List Content Type Read Only in SharePoint**

* REST-API is based on [**OData (Open Data) Protocol**](https://www.codesharepoint.com/interview-question/what-is-odata?) and hence it doesn't require any dll or JS library to run the commands on the SharePoint objects.
* REST-API calls require proper **end-point URL** and proper **Header** implementation in your REST call.

**How to use SharePoint REST API?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* You can also use this method in your **SPFx Component**.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

* Once you are done with setting your page, click on the button and your method should get executed.
* function SetListContentTypeReadOnly()
* {
* GetListContentTypeId();
* }
* function GetListContentTypeId()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url:  \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('List%20Name')/contenttypes?$filter=Name eq 'Content Type Title'&$top=1&$select=StringId",
* type: "GET",
* async:false,
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* },
* success: function (data, status, xhr) {
* // Content Type Id
* var dataresults = data.d.results;
* SetListContentTypeReadOnlyUsingContentTypeId(dataresults[0]["StringId"]);
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }
* function SetListContentTypeReadOnlyUsingContentTypeId(ContentTypeId)
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url:  \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('List%20Name')/contenttypes('" + ContentTypeId +"')",
* type: "POST",
* data: JSON.stringify({
* '\_\_metadata': {
* // Type that you are modifying.
* 'type': 'SP.ContentType'
* },
* 'ReadOnly': true
* }),
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "PATCH",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Set List Content Type Description in SharePoint using REST API**

……

* data: JSON.stringify
* ({
* '\_\_metadata': {
* // Type that you are modifying.
* 'type': 'SP.ContentType'
* },
* 'Description': 'Updated List Description'
* }),

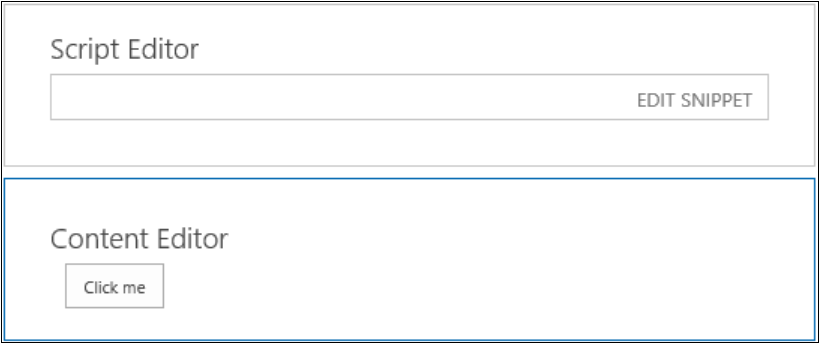
……..

**Enable Versioning in SharePoint using REST API**

The example in this topic show how to use REST API to Enable Versioning in SharePoint

* REST-API is based on [**OData (Open Data) Protocol**](https://www.codesharepoint.com/interview-question/what-is-odata?) and hence it doesn't require any dll or JS library to run the commands on the SharePoint objects.
* REST-API calls require proper **end-point URL** and proper **Header** implementation in your REST call.

**How to use SharePoint REST API?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* You can also use this method in your **SPFx Component**.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

* Once you are done with setting your page, click on the button and your method should get executed.
* function EnableVersioning() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('Documents')",
* // The HTTP request method: GET for read operations and POST for write operations.
* // POST requests can perform update or delete operations by specifying a DELETE, MERGE, or PUT verb in the X-HTTP-Method header.
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "MERGE",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* **EnableVersioning: true**
* }),
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Enable Minor Versioning in SharePoint using REST API**

……

* **data: JSON.stringify**
* **({**
* **\_\_metadata:**
* **{**
* **type: "SP.List"**
* **},**
* **EnableMinorVersions: true**
* **}),**

…..

**Enable Major Versioning in SharePoint using REST API**

……

* **data: JSON.stringify ({**
* **\_\_metadata:**
* **{**
* **type: "SP.List"**
* **},**
* **MajorVersionLimit: 55**
* **}),**

…..

**Enable Minor Versioning in SharePoint using REST API**

* data: JSON.stringify    ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* **EnableFolderCreation: true**
* }),

**Inherit Parent Permissions On List in SharePoint using REST API**

* function InheritParentPermissionOnList()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* // This method will work only if the role inheritence is broken(list has unique role assignments) on the list
* **url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web/lists/getByTitle('List%20Name')/ResetRoleInheritance()",**
* type: "POST",
* headers:
* {
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Enable On Quick Launch in SharePoint using REST API**

* function EnableOnQuickLaunch()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url:  \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('List%20Name')",
* type: "POST",
* data: JSON.stringify({
* '\_\_metadata': {
* // Type that you are modifying.
* 'type': 'SP.List'
* },
* **'OnQuickLaunch': true**}),
* headers: {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "PATCH",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Enabling Content Type**

* function AllowContentTypes() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('Documents')",
* // The HTTP request method: GET for read operations and POST for write operations.
* // POST requests can perform update or delete operations by specifying a DELETE, MERGE, or PUT verb in the X-HTTP-Method header.
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "MERGE",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* **AllowContentTypes: true,**
* **ContentTypesEnabled: true**
* }),
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**Add Content Type To List in SharePoint using REST API**

* function AddContentTypeToList()
* {
* GetSiteContentTypeId();
* }
* function GetSiteContentTypeId()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url:  \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/contenttypes?$filter=Name eq 'Content Type Title'&$top=1&$select=StringId",
* type: "GET",
* async:false,
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* },
* success: function (data, status, xhr) {
* var dataresults = data.d.results;
* AddSiteContentTypeToListUsingContentTypeId(dataresults[0]["StringId"]);
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }
* function AddSiteContentTypeToListUsingContentTypeId(ContentTypeId)
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url:  \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('List%20Name')/contenttypes/AddAvailableContentType",
* type: "POST",
* data: JSON.stringify({
* "contentTypeId": ContentTypeId}),
* headers: {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**NOTE:**

**success: function (data, status, xhr) { ... }**

This is the **callback function** that runs if the AJAX request succeeds.

* data: This contains the **response from SharePoint** (usually JSON).
* status: The status of the request (like "success").
* xhr: The full XMLHttpRequest object (not used here, but useful for debugging).

**var dataresults = data.d.results;**

This accesses the actual array of results returned by SharePoint.

💡 In SharePoint REST API responses:

* data.d is the top-level object.
* data.d.results is the array of items returned (e.g., a list of content types, list items, etc.).

So here, dataresults becomes an **array of objects** (each object is one content type).

**✅ DeleteContentTypeFromListUsingContentTypeId(dataresults[0]["StringId"]);**

Let’s break this in 2 parts:

1. dataresults[0] means:
   * Take the **first content type object** in the results array.
2. ["StringId"] means:
   * Access the **content type ID** of that first item (which is a string like "0x0100E...").
3. This full line calls a **custom function**:

js

CopyEdit

DeleteContentTypeFromListUsingContentTypeId("0x0100E...");

* + So this is deleting a content type from a list, **using its ID** (which SharePoint needs to identify it).

JSOM:

Creating list:

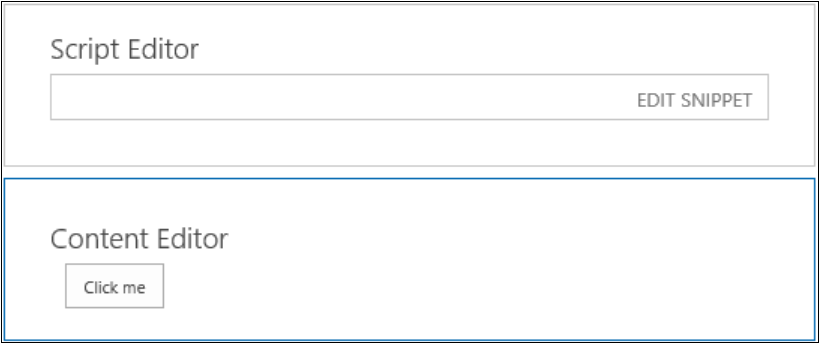
* function CreateList() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/",
* type: "POST",
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* Title: "List Name",
* Description: "New List Description",
* BaseTemplate: 100
* }),
* headers:
* {
* "Accept": "application/json;odata=verbose",
* "Content-Type": "application/json;odata=verbose",
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

Set List description in SharePoint using REST API

The example in this topic show how to use REST API to Set List description in SharePoint

* You can use **Content Editor** or **Script Editor web part**on your site, put **Html** and **JSOM (Java Script Object Model)** code and that's it. You can perform any action supported by JSOM APIs

**How to execute JSOM code in SharePoint?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. You can keep **jQuery reference** as per your need. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://code.jquery.com/jquery-2.2.4.js" type="text/javascript"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

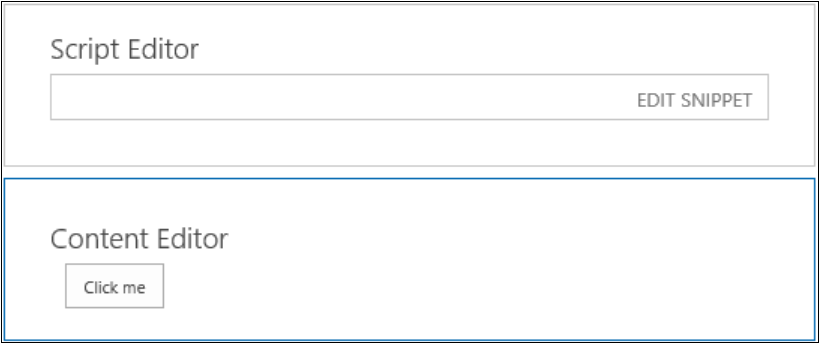
* Once you are done with setting your page, click on the button and your method should get executed.
* function SetListDescription() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('Documents')",
* // The HTTP request method: GET for read operations and POST for write operations.
* // POST requests can perform update or delete operations by specifying a DELETE, MERGE, or PUT verb in the X-HTTP-Method header.
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "MERGE",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* Description: "New List Description"
* }),
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });}

Enable Versioning in SharePoint using REST API

The example in this topic show how to use REST API to Enable Versioning in SharePoint

* You can use **Content Editor** or **Script Editor web part**on your site, put **Html** and **JSOM (Java Script Object Model)** code and that's it. You can perform any action supported by JSOM APIs

**How to execute JSOM code in SharePoint?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. You can keep **jQuery reference** as per your need. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://code.jquery.com/jquery-2.2.4.js" type="text/javascript"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

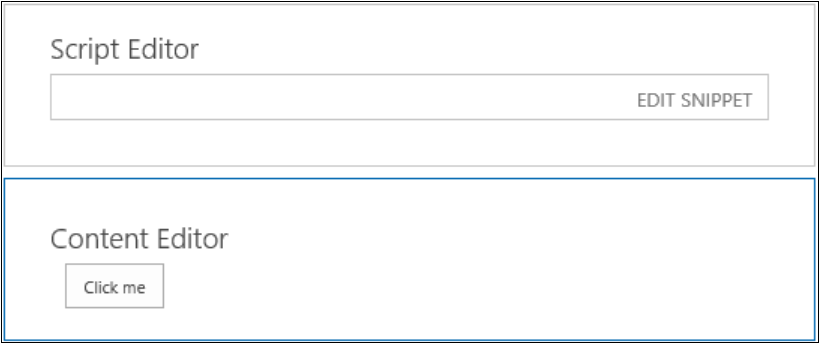
* Once you are done with setting your page, click on the button and your method should get executed.
* function EnableVersioning() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('Documents')",
* // The HTTP request method: GET for read operations and POST for write operations.
* // POST requests can perform update or delete operations by specifying a DELETE, MERGE, or PUT verb in the X-HTTP-Method header.
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "MERGE",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* EnableVersioning: true
* }),
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

Enable Folder Creation in SharePoint using REST API

The example in this topic show how to use REST API to Enable Folder Creation in SharePoint

* You can use **Content Editor** or **Script Editor web part**on your site, put **Html** and **JSOM (Java Script Object Model)** code and that's it. You can perform any action supported by JSOM APIs

**How to execute JSOM code in SharePoint?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. You can keep **jQuery reference** as per your need. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://code.jquery.com/jquery-2.2.4.js" type="text/javascript"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

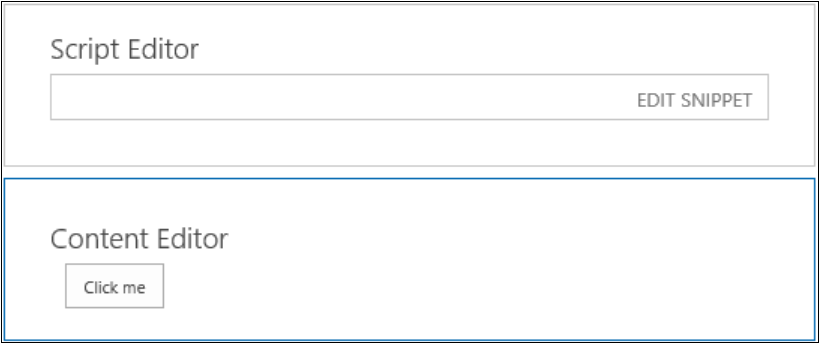
* Once you are done with setting your page, click on the button and your method should get executed.
* function EnableFolderCreation() {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('Documents')",
* // The HTTP request method: GET for read operations and POST for write operations.
* // POST requests can perform update or delete operations by specifying a DELETE, MERGE, or PUT verb in the X-HTTP-Method header.
* type: "POST",
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose",
* //Content-Type header: Specifies the format of the data that the client is sending to the server
* "Content-Type": "application/json;odata=verbose",
* // IF-MATCH header: Provides a way to verify that the object being changed has not been changed since it was last retrieved.
* // "IF-MATCH":"\*", will overwrite any modification in the object, since it was last retrieved.
* "IF-MATCH": "\*",
* //X-HTTP-Method:  The MERGE method updates only the properties of the entity , while the PUT method replaces the existing entity with a new one that you supply in the body of the POST
* "X-HTTP-Method": "MERGE",
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* data: JSON.stringify
* ({
* \_\_metadata:
* {
* type: "SP.List"
* },
* EnableFolderCreation: true
* }),
* success: function (data, status, xhr) {
* console.log("success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

Inherit Parent Permissions On List in SharePoint using REST API

The example in this topic show how to use REST API to Inherit Parent Permissions On List in SharePoint

* You can use **Content Editor** or **Script Editor web part**on your site, put **Html** and **JSOM (Java Script Object Model)** code and that's it. You can perform any action supported by JSOM APIs

**How to execute JSOM code in SharePoint?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. You can keep **jQuery reference** as per your need. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://code.jquery.com/jquery-2.2.4.js" type="text/javascript"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

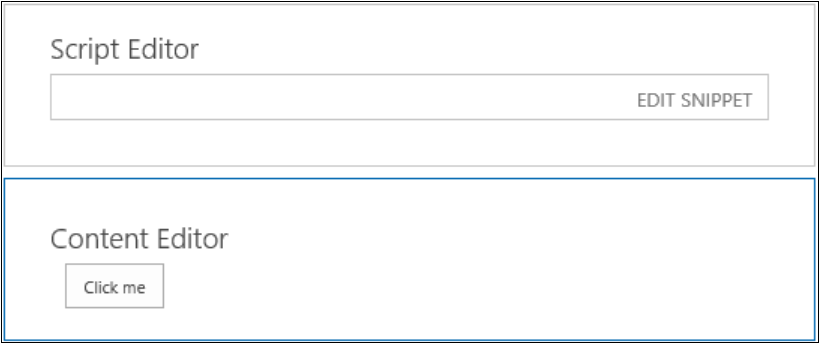
* Once you are done with setting your page, click on the button and your method should get executed.
* function InheritParentPermissionOnList()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* // This method will work only if the role inheritence is broken(list has unique role assignments) on the list
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/Web/lists/getByTitle('List%20Name')/ResetRoleInheritance()",
* type: "POST",
* headers:
* {
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });       }

Add Group To List in SharePoint using REST API

The example in this topic show how to use REST API to Remove Groups Permissions From List in SharePoint

* You can use **Content Editor** or **Script Editor web part**on your site, put **Html** and **JSOM (Java Script Object Model)** code and that's it. You can perform any action supported by JSOM APIs

**How to execute JSOM code in SharePoint?**

* **Note**: This is just a sample, you can go ahead with your implementation.
* Open your Site. Create a **webpart page**.
* Put a **Content Editor Web part** and a **Script Editor Web part** on it.
* I have added below **Html** in **Content Editor webpart**. You can replace MainFunction with your actual method name.

<button onclick="MainFunction();" type="button">Click me​</button>

* And below **references** in **Script Editor Webpart**. You can keep **jQuery reference** as per your need. **ScriptFile.js** is a JavaScript file where you can keep below function **i.e.** your actual logic.

<script src="https://code.jquery.com/jquery-2.2.4.js" type="text/javascript"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script src="https://MyServer/sites/SiteCollection/style library/js/ScriptFile.js"></script>

* Once you are done with setting your page, click on the button and your method should get executed.
* function AddGroupToList()
* {
* /\* Functions to perform in sequence:
* 1) Get Group ID from Site - GetGroupId()
* 2) Get RoleDefinition ID - GetRoleDefinitionId()
* 3) Use Group ID and RoleDefinition ID to perform action - AddGroupToListUsingGroupIdAndRoleDefinitionId() \*/
* GetGroupId();
* }
* function GetGroupId()
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/sitegroups/getbyname('Designers')/id",
* type: "GET",
* async: false,
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose"
* },
* success: function (data, status, xhr) {
* var GroupId = data.d.Id;
* GetRoleDefinitionId(GroupId)
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }
* function GetRoleDefinitionId(GroupId)
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/roledefinitions/getbyname('Read')/id",
* type: "GET",
* async: false,
* headers:
* {
* // Accept header: Specifies the format for response data from the server.
* "Accept": "application/json;odata=verbose"
* },
* success: function (data, status, xhr) {
* var RoleDefinitionId = data.d.Id;
* AddGroupToListUsingGroupIdAndRoleDefinitionId(GroupId,RoleDefinitionId);
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }
* function AddGroupToListUsingGroupIdAndRoleDefinitionId(GroupId,RoleDefinitionId)
* {
* $.ajax
* ({
* // \_spPageContextInfo.webAbsoluteUrl - will give absolute URL of the site where you are running the code.
* // You can replace this with other site URL where you want to apply the function
* url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('List%20Name')/roleassignments/addroleassignment(principalid='" + GroupId + "',roledefid='" + RoleDefinitionId +"')",
* type: "POST",
* headers:
* {
* // X-RequestDigest header: When you send a POST request, it must include the form digest value in X-RequestDigest header
* "X-RequestDigest": $("#\_\_REQUESTDIGEST").val()
* },
* success: function (data, status, xhr) {
* console.log("Success");
* },
* error: function (xhr, status, error) {
* console.log("Failed");
* }
* });
* }

**DIFF BETWEEN JSOM AND REST API (THOERY AND CODE):**

Reasoned for a couple of seconds

In SharePoint 2016 on-premise, **JSOM is a good choice** because:

* **Native Support**: It’s built into SharePoint, so it’s fully supported and integrated.
* **Rich SharePoint Object Model**: You get direct access to SharePoint objects, methods, and properties that simplify complex operations.
* **Simpler Setup**: No need for extra configuration like CORS or custom authentication headers that REST might require.
* **Legacy Compatibility**: It's well-tested and works reliably in older SharePoint versions where REST support might be more limited.

2/2

o3-mini

**You said:**

what is conding part diff be specific and ashor

**ChatGPT said:**

**🆚 Coding Difference: JSOM vs REST (Short & Specific)**

**✅ JSOM (JavaScript Object Model)**

* **Style**: Object-based
* **Example**:

javascript

CopyEdit

var context = new SP.ClientContext.get\_current();

var list = context.get\_web().get\_lists().getByTitle('MyList');

var items = list.getItems(SP.CamlQuery.createAllItemsQuery());

context.load(items);

context.executeQueryAsync(function() {

alert("Success");

}, function(sender, args) {

alert("Error: " + args.get\_message());

});

**✅ REST API**

* **Style**: URL-based, AJAX calls
* **Example**:

javascript

CopyEdit

$.ajax({

url: \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/getByTitle('MyList')/items",

type: "GET",

headers: {

"Accept": "application/json;odata=verbose"

},

success: function(data) {

alert("Success");

},

error: function(error) {

alert("Error");

}

});

**⚖️ Summary:**

* **JSOM** → More **SharePoint-like** code, good for complex SharePoint logic.
* **REST** → More **standard web code**, better for integration and modern use.