S-Drive
Developer Guide
v2.7

Important Note

This user guide contains detailed information about S-Drive customization with special APIs and intended for developer use. Refer to the *S-Drive Admin Guide and S-Drive User Guide* for more information about installation/configuration and usage of S-Drive product.

CyanGate

www.cyangate.com

Contents

I. SDriveTools API

You can use SDriveTools API calls for programmatically interacting with S-Drive Attachments.

A. getAttachmentURL()

This method is used to get the URL of an S-Drive Attachment.

String getAttachmentURL(String parentId, String fileObjectId,
Long timeValue)

Parameters:

parentId: Id of the parent object. You can use 15-character id or 18-character id.

fileObjectId: Id of the attachment (file) object. This can be retrieved using an SOQL query. An example can be found in following sections.

timeValue: Expiration time of the link in seconds.

Return Value: The method will return the URL of the S-Drive attachment.

Our example is about retrieving an image file from S-Drive Account Attachments and displaying it inside Account Page Layout. You can customize SOQL call and other options similar to the example.

1. First override Accounts View button with Account Files Page to upload a JPG image file to S-Drive Account Attachments. (Figure 1)

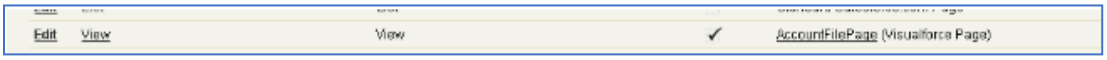


Figure 1

2. Upload a jpg image file into one of your accounts using Upload File(s) button. (Figure 2)

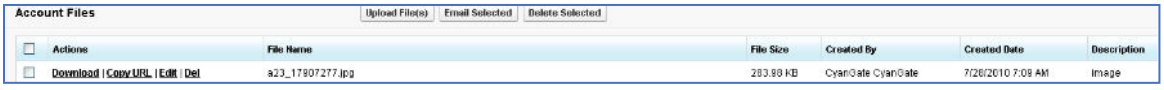


Figure 2

3. Create an apex class named ExamplePageController. In below code 1*60 is equivalent to one minute. You can set expiring time like this. Also, you can configure your SOQL query based on your needs. (Figure 3)

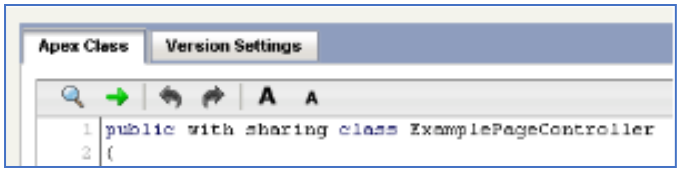


Figure 3

```
public with sharing class ExamplePageController
{
    private Account acct;
    private String fileURL = '';

    public ExamplePageController(ApexPages.StandardController
controller)
    {
        this.acct = (Account)controller.getRecord();

        List<cg__AccountFile__c> accountFiles = [Select id from
cg__AccountFile__c where cg__AccountFile__c.cg__WIP__c = false and
cg__AccountFile__c.cg__Content_Type__c = 'image/jpg' and
cg__AccountFile__c.cg__Account__c = :acct.id];

        if(accountFiles.size() > 0)
        {
            fileURL = cg.SDriveTools.getAttachmentURL(acct.id,
accountFiles[0].id, (1 * 60));
        }
        else
        {
            fileURL = 'http://www.cyangate.com/noimage.jpg';
        }
    }
    public String getFileURL()
    {
        return fileURL;
    }
}
```

4. Create an ExamplePage apex page with below content. (Figure 4)

Figure 4

```
<apex:page standardController="Account"
extensions="ExamplePageController">
      <apex:image url="{!fileURL}" />
</apex:page>
```

5. Customize your account page layout and include ExamplePage inside the layout. (Figure 5)

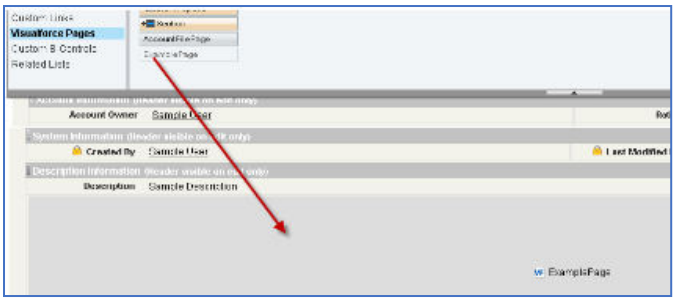


Figure 5

6. Now if you reload your account (which contains an image file) you'll see a screen like below. You can customize the apex class and page based on your needs. (Figure 6)

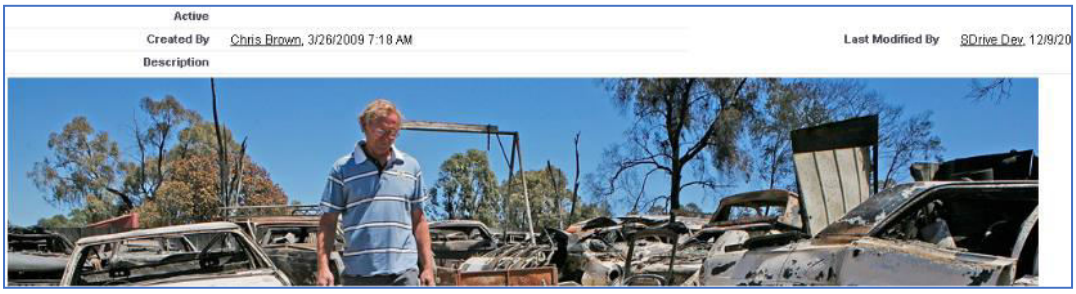


Figure 6

Notes

1. If you would like to get an old version of a file, you should add id of an old version file into fileoObjectIds list.

B. getAttachmentURLs()

This method is used to get multiple URLs for multiple objects of an S-Drive Attachment at a time.

```
List<String> getAttachmentURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue)

List<String> getAttachmentURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue, Map<String,String> requestParameters)

List<String> getAttachmentURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue, List<Map<String,String>>
   requestParametersList)
```

Parameters:

parentIds: List of Salesforce ids of the parent objects. You can use 15-character id or 18-character id.

fileObjectIds: List of Salesforce ids of the attachment (files) objects.

timeValue: Expiration time of the links in seconds.

requestParameters: Map of request parameters and theirs values to set the parameters in the response (e.g. (*'response-content-disposition'*, *'inline; filename=myfile.png'*)). For more information, visit the Amazon documentation:

http://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectGET.html

requestParametersList: List of request parameters map for each files.

Return Value: The method will return the list of URLs of the S-Drive attachment.

Notes

1. If you would like to get an old version of a file, you should add id of an old version file into fileoObjectIds list.

C. id15to18()

This method is used to convert Salesforce Ids from 15 characters to 18 characters.

String id15to18(String inID)

Parameters:

inID: 15-character Salesforce id.

Return Value: The method will return the 18-character Salesforce id.

D. getAccessKey()

This method is used to get the AccessKey for setting the *AWSAccessKeyId* parameter during uploads.

```
String getAccessKey()
```

E. getBucketName()

This method is used to get the Bucket Name for setting the S3 *bucket* parameter during uploads.

```
String getBucketName()
```

F. getS3Endpoint()

This method is used to get the S3 Endpoint location of your S3 bucket for setting the S3 *endpoint* parameter during uploads.

```
String getS3Endpoint()
```

G. deleteFiles()

This method is used to delete files stored as attachments or under S-Drive folders.

```
List<ResultObject> deleteFiles(List<ID> wipIds, String objectId)
```

Parameters:

wipIds: List of Salesforce.com IDs of "file" records (either attachments or S-Drive Folder files).

objectId: Id of the parent object. You can use 15-character or 18-character Salesforce.com ID.

Return Value: The method will return a list of ResultObject (Figure 8) which holds delete status information for each file record.

Example code:

```
List<Id>wipIds = new List<Id>();

wipIds.add(ID.valueOf(uploadRequestInfos .fileWipId));

List<cg.ResultObject>resultObject=cg.SDriveTools.deleteFiles(
                                        wipIds, objectId);
```

Notes

1. If you enable versioning and add a latest version file id into wipIds list, all versions of that file will also be deleted. You can also delete a single old version file by adding its id into wipId list.

H. initializeUpload()

This method is used to initialize attachment uploads.

```
List<UploadRequestInfo> initializeUpload
(String objectId, List<SObject> attachments, Map<String,String> policyMap)
```

Parameters:

objectId: Id of the parent object. You can use 15-character or 18-character Salesforce.com id. For example, this ID is the ID of the case record if the attachments are being uploaded for a case.

attachments: List of Salesforce SObject for uploading. The SObject will be representing the "File" object. For example, for a Case attachment, the SObject will be representing the cg__CaseFile__c record.

policyMap: Map of policy conditions and theirs values that represent the additional policy parameters used during upload (e.g. (*'$Content-Disposition'*, '*attachment; filename*)). For more information, visit the Amazon documentation:

http://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectPOST.html

Return Value: The method will return the list of UploadRequestInfo (Figure 7), which holds information about the files that are about to be uploaded.

The following steps are executed within this method:

a. Validate the input with the following rules:
   If the uploaded attachments are of type S3Object (i.e. files in S-Drive tab), the Parent__c field should be the same for all files.
b. If the uploaded attachments are of type S3Object (i.e. files in S-Drive tab), there should not be an existing file with the same name in the same folder.
c. File name and file size can't be blank
d. File name can't include the following characters: \ / : * ? \" <> | ~
e. File size can't be zero.
f. File size can't be more than the Max file size defined in S-Drive Configuration.

1. Create the file record in Salesforce with work in progress state (WIP__c = true) (i.e. create S3Object, Case File, Custom Object File, etc).
2. Calculate policy, signature, file name etc. in order to be used as the POST parameters. While calculating the policy, policyMap parameter is used to generate different parameters in the policy. For html upload please reference Figure 9 and Returns a list of UploadRequestInfo (Figure 7) objects.

Example code:

```
List<SObject>attachments = new List<SObject>();

My_Example_Object_File__c attachment = new My_Example_Object_File__c
();

attachment.File_Name__c = 'Example.txt';

attachment.File_Size_in_Bytes__c ='100';

attachment.Parent__c = 'a0I80…';

String objectId = String.valueOf(attachment.Parent__c);

attachments.add(attachment);

Map<String, String> policyMap= new Map<String,String>();

policyMap.put('$Content-Disposition','attachment; filename');

List<cg.UploadRequestInfo> uploadRequestInfos=

cg.SDriveTools.initializeUpload(objectId ,attachments, policyMap );
```

Notes

1. It is required to set SObject's File_Name__c, File_Size_in_Bytes__c and Parent__c fields. You can optionally set other custom fields.
2. For S-Drive Attachments files, Parent__c refers to its parent object id. In order to set the parent folder id, use Parent_Folder_Id__c field. For S-Drive Folders files, Parent__c refers to its parent folder id and if you want to upload the file to home folder set this to **null**. Also for S-Drive Folders files, objectId must to be set to **null**.
3. If you put expiration, *acl*, *bucket*, *key* and x-*amz-server-side-encryption conditions* in the **policyMap**, these parameters will be ignored. Because, these are being set in the initializeUpload () method. You do not need to put these parameters in the **policyMap**. But, you must use these parameters while uploading. For this, please reference Figure 10.

I. completeUpload()

This method is used to complete attachments upload once the files have been uploaded to Amazon S3.

List<ResultObject> completeUpload(List<ID> wipIds)

Parameters:

wipIds: List of Salesforce ids for attachment "file" records. These IDs have been returned from the initializeUpload() method for each file record.

Return Value: The method will return the list of Result Object (Figure 127), which holds filecompletion status information.

Notes

1. If you want to confirm that file(s) are successfully uploaded to Amazon, you can use the *getAmazonHeader()* method to get headers which is returned by Amazon S3 for a given file. You can also compare the *ETag* header, the MD5 checksum calculated by Amazon when the file is uploaded to S3, with the MD5 that you calculate for the file(s).
2. If you set the success_*action_status* to *201* status code during the upload process, you can also use XML document which is returned by Amazon S3 to compare *ETag* header with MD5 check sum.

Example code:

```
List<Id>wipIds = new List<Id>();

wipIds.add(ID.valueOf(uploadRequestInfos .fileWipId));

 List<cg.ResultObject>resultObjects = cg.SDriveTools.completeUpload(wipIds);
```

J. cancelUpload()

This method is used to cancel attachment upload operation.

List<ResultObject> cancelUpload(List<ID> wipIds, String objectId)

Parameters:

wipIds: List of Salesforce.com IDs of attachment "file" records.

objectId: Id of the parent object. You can use 15-character or 18-character Salesforce.com ID.

Return Value: The method will return a list of ResultObject (Figure 127) which holds cancel status information for each file record.

Example code:

```
List<Id>wipIds = new List<Id>();

wipIds.add(ID.valueOf(uploadRequestInfos .fileWipId));

List<cg.ResultObject>resultObjects =
cg.SDriveTools.cancelUpload(wipIds, objectId);
```

K. getAmazonHeaders()

This method is intended to be used to get response headers and its values from Amazon after an upload in order to verify the successful upload of the files.

```
Map<String,String> getAmazonHeaders(String item)
```

Parameters:

item: File location/key of uploaded file.

Return Value: The method will return the map of response headers and its values (e.g. ETag)

Example code:

```
Map<String,String> headResponse =
cg.SDriveTools.getAmazonHeaders(uploadRequestInfos .fileLocation);
```

L. initializeMultiPartUpload()

This method is used to initialize multipart upload to get uploadId. This uploadId will be used on copy part, complete, abort multi part operations.

```
String initializeMultiPartUpload(String awsLocation)
```

Parameters:

awsLocation: The name of key to be uploaded as file location. You can get this value from the fileLocation of UploadRequestInfo object.

Return Value: The method will return the uploadId as a String.

Example code:

```
String uploadId =
cg.SDriveTools.initializeMultiPartUpload(uploadRequestInfos.fileLocatio
n);
```

M. copyPartMultiPartUpload()

This method is used to upload a part by copying data from existing object as data source to get Etag value. This Etag will be used complete multi part request.

```
String copyPartMultiPartUpload(String awsLocation, String uploadId, Long
partNumber)
```

Parameters:

awsLocation: The name of key to be uploaded as file location. You can get this value from the fileLocation of UploadRequestInfo object.

uploadId: The upload Id that you get this Id from the initializeMultipartUpload request as a return value.

partNumber: The order of uploaded part.

Return Value: The method will return the Etag as a String.

Example code:

```
List<String> eTagList = new List <String>();
for(Integer i = 1; i<= multiPartsSize; i++)
{
     String eTag = cg.SDriveTools.copyPartMultiPartUpload
                    (uploadRequestInfos .fileLocation, uploadId, i);
        eTagList.add(eTag);
}
```

Notes

1. While uploading multi parts, you must give the *key* parameters with below format. It must be start with zero.

"key" : *fileLocation* + '.' +0 (partNumber = 1)
to
"key" : *fileLocation* + '.' + multiPartsSize-1 (partNumber = multiPartsSize)

N. completeMultiPartUpload()

This method is used to complete a multipart.

```
String completeMultiPartUpload(String awsLocation, String uploadId, List<String>
eTagList)
```

Parameters:

awsLocation: The name of key to be uploaded as file location. You can get this value from the fileLocation of UploadRequestInfo object.

uploadId: The upload Id that you get this Id from the initializeMultipartUpload request as a return value.

eTagList: List of Etags. You can get Etags by copyPartMultiPart requests.

Return Value: The method will return the ETag as a String.

Example code:

```
String ETag = cg.SDriveTools.completeMultiPartUpload(uploadRequestInfos
.fileLocation, uploadId,  eTagList);
```

O. deleteMultiParts()

This method is used to delete multi parts.

```
void deleteMultiParts(String awsMultiPartsLocationList)
```

Parameters:

awsMultiPartsLocationList: List of parts keys as the names of file locatons. It must contain up to 1000 keys because of limit.

Example code:

```
List<String> awsMultiPartsLocationList = new List <String>();
for(Integer i = 0; i<= multiPartsSize-1; i++)
{
   awsMultiPartsLocationList.add( fileLocation + '.' +i );
}

cg.SDriveTools.deleteMultiParts(awsMultiPartsLocationList);
```

P. abortMultiPartUpload()

This method is used to abort a multipart upload.

```
void abortMultiPartUpload(String awsLocation, String uploadId)
```

Parameters:

awsLocation: The name of key to be uploaded as file location. You can get this value from the fileLocation of UploadRequestInfo object.

uploadId: The upload Id that you get this Id from the initializeMultipartUpload request as a return value.

Example code:

```
cg.SDriveTools.abortMultiPartUpload(uploadRequestInfos
.fileLocation, uploadId);
```

Q. inheritSharings()

This method is used to inherit the sharings from parent folder to the file.

```
void inheritSharings(List<String> wipIdList, String currentFolderId)
```

Parameters:

wipIdList: List of Salesforce.com IDs of "file" records(either attachments or S-Drive Folder files) as a String.

currentFolderId: Id of the parent folder of the file for sharing.

Example code:

```
List<String> wipIdList = new List <String>();
wipIdList .add(uploadRequestInfos .fileWipId);
cg.SDriveTools.inheritSharings(wipIdList,  currentFolderId);
```

UploadRequestInfo
String fileName{get; set;} *String fileLocation{get; set;}*

```
String fileSize{get; set;}
String fileType{get; set;}
String wipFileId{get; set;}
String signature{get; set;}
 String policy{get; set;}
```

Figure 7

ResultObject

`String status {get;set;}//` *success – fail*
`String errorMessage {get;set;} //` *null if status is success*
`String wipFileId {get;set;} //` *to see which file failed*

Figure 8

R. Uploading Files to S-Drive (Amazon S3)

You can use SDriveTools API calls for uploading the S-Drive Attachments/Folders Files to Amazon. For this, you will need to use below methods:

- `initializeUpload()`
- `completeUpload()`
- `cancelUpload()`
- `getAccessKey()`
- `getBucketName()`
- `getS3Endpoint()`

Before upload, you must first initialize attachment upload. Initializing the attachment upload creates work in progress (WIP) file(s) objects (S3Object, Case File, My Example Object File, etc.) and calculates the required file information for the upload process. For initializing attachment upload, use initializeUpload() method.

The next step is uploading the file(s) to Amazon using html upload. When you upload file(s) to Amazon, You should use the information which has been returned by initializeUpload() as well as the values retrieved from getAccessKey(), getBucketName() and getS3Endpoint() methods.

If upload is successful, you must complete attachment upload. Complete attachment upload means updating WIP file records created during initialize upload call and set the *WIP__c* field value as *false*. For completing attachment upload, use completeUpload() method.

If upload fails, you must cancel upload. Canceling upload means deleting the WIP file records. For canceling attachment upload, use cancelUpload() method.

You can also upload the big files using multipart uploads. For this, you will need below methods.

- `initializeMultiPartUpload()`
- `copyPartMultiPartUpload()`
- `completeMultiPartUpload()`
- `deleteMultiParts()`
- `abortMultiPartUpload()`

Html Policy

```
{
"expiration": "2014-03-12T12:47:13.893Z",
"conditions":
[
{"acl": "private" },
{"bucket": "cg--81..." },
["starts-with", "$Content-Disposition", "attachment; filename"],
["starts-with", "$key", "a038…"],
{"x-amz-server-side-encryption": "AES256"},
]
}
```

Figure 9

1. Exceptions and Reasons for Upload Operations

1. **[SDriveException]** *"Parent Folder IDs must be the same for all file records that are being uploaded".*
   [Reason] If Parent__c values which are passed in with the SObjects are not same for all S3Object.
   [Solution] Give the parent folder ids the same for all S3Objects.

2. **[SDriveException]** *"A file with the same name exists in the target folder (fileName) ".*
   [Reason] This exception occurs if a file with the same name exists in the same folder for S3Objects.The same file name condition holds when the file is not a WIP (WIP__c =false) file or the file is a WIP file (WIP__c =true) with the same file namebutithasbeenuploadedbyanotheruserlessthan12 hours ago.
   [Solution]Upload files with different file names or delete the previously existing files before the upload.

3. **[SDriveException]** *"Error occurred while checking the files to upload! File Name, File Size cannot be blank (fileName)".*
   [Reason]If the file name or file size fields that have been passed in with the SObject are null or empty.
   [Solution]File name and file size is required for upload. Thus, pass non-blank values in with the SObject.

4. **[SDriveException]** *"Name cannot start with a space or a dot and cannot contain any of the following characters: \\ /: *? \" < > | ~ (fileName)".*
   [Reason]If the file name is invalid. Invalid name means, file name starts with a spaceor a dot and contains any of the following characters: *\ / : * ? \" < > | ~*
   [Solution]Do not upload a file with the name starting with space or a dot and contain any of the following characters: *\ / : * ? \" < > | ~*

5. **[SDriveException]** *"You cannot upload a zero-length file (fileName)".*
   [Reason] If the file size is 0.
   [Solution] Upload files that are greater in size than zero.

6. **[SDriveException]** *"Files greater than (maxFileSize) cannot be uploaded. Please contact your system administrator for the file size limits! (fileName)".*
   [Reason]File being uploaded has a size greater than the maximum file size limit (MAX_FILE_SIZE configuration in S-Drive Configuration page).
   [Solution]Do not upload files whose size greater than MAX_FILE_SIZE. Or Increase the MAX_FILE_SIZE configuration in S-Drive Configuration page.

7. **[SDriveException]**"*Invalid parent id specified. Check your function parameters!".*
   [Reason] If object (parent) id, which is passed as a parameter, is not a Salesforce.com 15 or 18 characters long ID.
   [Solution]Pass 15-character or 18-characterSalesforce.com ID as objectId parameter.

2. Error Messagesof ResultObject and Reasons

1. **[errorMessage]***"Wip Id is null! ".*
   [Reason] If wipId in the wipIds list parameter is null.
   [Solution] Do not pass null Salesforce.com IDs of file objects.

2. **[errorMessage]***"'No such wip file with provided id: (wipId) ".*
   [Reason]If wipId in wipIds list is not available.
   [Solution]Pass correct and available Salesforce.com ids of files objects.

3. **[errorMessage]***"There is no wip files".*
   [Reason]If all of wipId in wipIds list is null or wipIds' size is 0.
   [Solution]Pass wipIds with length greater than0 and non-null string values.

4. **[errorMessage]***" Insufficient Privileges. You do not have not access to update operation".*
   [Reason]If Object-level security for update access is not allowed.
   [Solution] Make sure that update access is allowed for your profile.

   Html upload example:

```
<form>
File : <input type="file" name="file" id="file" />
      <input type="button" value="Upload" onclick="htmlUpload();"/>
</form>

<script type="text/javascript">

var uploadFile ;
document.getElementById('file').addEventListener('change',
handleFileSelect,false);

function handleFileSelect(evt)
{
      var fileName = evt.currentTarget.files[0].name ;
      var fileSize = evt.currentTarget.files[0].size;
      uploadFile = evt.currentTarget.files[0];
      //Call initializeUpload() method
}
```

```
function htmlUpload()
{
	var fd = new FormData();
	fd.append('key', '{!uploadRequestInfos.fileLocation}');
	fd.append('AWSAccessKeyId', '{!accessKey}');
	fd.append('acl', 'private');
	fd.append('policy', '{!uploadRequestInfos.policy}');
	fd.append('signature', '{!uploadRequestInfos.signature}');
	fd.append('x-amz-server-side-encryption', 'AES256');
	fd.append('Content-Disposition', 'attachment; filename=\"'+
'{!uploadRequestInfos.fileName}' +'\"');
	fd.append("file",uploadFile);

	var xhr = new XMLHttpRequest();
	xhr.addEventListener("load", htmlUploadComplete, false);
	xhr.addEventListener("error", htmlUploadFailed, false);
	xhr.open('POST', 'https://s3.amazonaws.com/'+'{!bucketName}',
true);

	xhr.send(fd);
}


function htmlUploadComplete(evt)
{
	  //Call completeUpload()  method
}

function htmlUploadFailed(evt)
{
	  //Call cancelUpload() method
}
</script>
```

You can see the appropriate values for the request parameters below (Figure 10).

Form Field	Value
acl	'private'
AWSAccessKeyId	*SDriveTools.getAccessKey()*
bucket	*SDriveTools.getBucketName()*
Content-Disposition	'attachment; filename=\"'+ uploadRequestInfos.*fileName*+'\"'
Content-Type	uploadRequestInfos.*fileType*

key *policy* *signature* *x-amz-server-side-encryption* *success_action_status*	uploadRequestInfos.*fileLocation* uploadRequestInfos.*policy* uploadRequestInfos.*signature* 'AES256' 201

Figure 10

S. getPreviewURL()

This method is used to get the URL of an S-Drive Attachment's Preview File.

```
String getPreviewURL(String parentId, String fileObjectId,
     Long timeValue)
```

Parameters:

parentId: Id of the parent object. You can use 15-character id or 18-character id.

fileObjectId: Id of the attachment (file) object. This can be retrieved using an SOQL query.

timeValue: Expiration time of the link in seconds.

Return Value: The method will return the URL of the S-Drive attachment' preview.

Notes

1. If you would like to get preview URL of an old version of a file, you should add id of an old version file into fileoObjectIds list.

T. getPreviewURLs()

This method is used to get multiple preview URLs for multiple objects of an S-Drive Attachment's at a time.

```
List<String> getPreviewURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue)

List<String> getPreviewURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue, Map<String,String> requestParameters)
```

```
List<String> getPreviewURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue, List<Map<String,String>>
   requestParametersList)
```

Parameters:

parentIds: List of Salesforce ids of the parent objects. You can use 15-character id or 18-character id.

fileObjectIds: List of Salesforce ids of the attachment (files) objects.

timeValue: Expiration time of the links in seconds.

requestParameters: Map of request parameters and theirs values to set the parameters in the response (e.g. (*'response-content-disposition'*, *'inline; filename=myfile.png'*)). For more information, visit the Amazon documentation:

http://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectGET.html

requestParametersList: List of request parameters map for each files.

Return Value: The method will return the list of preview URLs of the S-Drive attachment.

Notes

1. If you would like to get preview URL of an old version of a file, you should add id of an old version file into fileoObjectIds list.

U. getThumbnailURL()

This method is used to get the URL of an S-Drive Attachment's Thumbnail File.

```
String getThubmnailURL(String parentId, String fileObjectId,
     Long timeValue)
```

Parameters:

parentId: Id of the parent object. You can use 15-character id or 18-character id.

fileObjectId: Id of the attachment (file) object. This can be retrieved using an SOQL query.

timeValue: Expiration time of the link in seconds.

Return Value: The method will return the URL of the S-Drive attachment's thumbnail.

Notes

1. If you would like to get thumbnail URL of an old version of a file, you should add id of an old version file into fileoObjectIds list.

V. getThumbnailURLs()

This method is used to get multiple thumbnail URLs for multiple objects of an S-Drive Attachment at a time.

```
List<String> getThumbnailURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue)

List<String> getThumbnailURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue, Map<String,String> requestParameters)

List<String> getThumbnailURLs(List<ID> parentIds, List<ID>
  fileObjectIds, Long timeValue, List<Map<String,String>>
   requestParametersList)
```

Parameters:

parentIds: List of Salesforce ids of the parent objects. You can use 15-character id or 18-character id.

fileObjectIds: List of Salesforce ids of the attachment (files) objects.

timeValue: Expiration time of the links in seconds.

requestParameters: Map of request parameters and theirs values to set the parameters in the response (e.g. (*'response-content-disposition'*, *'inline; filename=myfile.png'*)). For more information, visit the Amazon documentation:

http://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectGET.html

requestParametersList: List of request parameters map for each file.

Return Value: The method will return the list of thumbnail URLs of the S-Drive attachment.

Notes

1. If you would like to get thumbnail URL of an old version of a file, you should add id of an old version file into fileoObjectIds list.

W. createFileActivity()

You can use SDriveTools API class to create file activities, after configuring the file activities from S-Drive configuration tab.

In order to create file activities using API class call the following methods depending on single file or multiple files:

cg.SDriveTools.createFileActivity(fileId, activityType,prefix,objectName,additionalDetails);

The method signature is explained as follows:

Create file activity record for the given file.

@param String fileId Id of the file to record its activity

@param String activityType type of activity

@param String prefix prefix of the file object e.g. 'cg__'

@param String objectName Name of file object which fileId belongs to e.g. AccountFile__c

@param String additionalDetails Additional details for file activity

throws SDriveException throws exception if fileId is empty or File Activity record insertion fails

global static void createFileActivity(String fileId, String activityType,String prefix, String objectName, String additionalDetails)

cg.SDriveTools.createFileActivity(fileId, activityType,prefix,objectName,additionalDetails);

The method signature is explained as follows:

Create file activity record for multiple files.

@param List<String> fileIds the file Ids to record their activity

The rest of the parameters are the same for two methods.

global static void createFileActivity(List<String> fileIds, String activityType, String prefix, String objectName, String additionalDetails)

X. createShortcuts()

This method is used to create shortcuts of files.

```
void createShortcuts(String destinationObjectFilePrefix, String
destinationObjectFileName, String destinationRelationshipName, String
destinationObjectId, String sourceObjectFilePrefix, String
sourceObjectFileName, String selectedFolder, List<String> fileIds)
```

Parameters:

destinationObjectFilePrefix: This is the object's prefix where you want to create the shortcuts. For S-Drive file objects included with the managed package, provide "cg__" for this parameter.

destinationObjectFileName: This is the object's name where you want to create the shortcuts.

destinationRelationshipName: This is the relationship field's name between the file object and its parent. For your custom objects provide "Parent__r", for objects included with S-Drive managed package, provide the Master-Detail relationship or Lookup relationship name.

destinationObjectId: The parent record's ID where you want shortcuts to be created.

sourceObjectFilePrefix: The prefix of the object of the source/original files that shortcuts will be created from.

sourceObjectFileName: The name of the object of the source/original files that shortcuts will be created from.

selectedFolder: If you want to put the shortcuts into a specific folder in the destination, provide the ID of the folder that you created in the target record's S-Drive.

fileIds: The source/original files' IDs in a List<String> format.

Return Value: The method does not return anything.

Example code:

The example code will create shortcuts as Account File records and put them under the specified Account which has the accountID. The shortcuts will be created from Case Files which have caseFileIds.

```
String accountID;

List<String> caseFileIds;

cg.SDriveTools.createShortcuts('cg__', 'AccountFile__c', 'Account__r',
accountID, 'cg__', 'CaseFile__c', '', caseFileIds);
```

Y. createPublicDownloadLink()

You can use SDriveTools API class to create a public download link, after configuring S-URL from S-Drive Configuration tab. To create a public download link using API class call the following method:

```
cg.SDriveTools.createPublicDownloadLink(fileIds, objectFileName,
expirationDate, password, IpRange);
```

The method signature is explained as follows:

@param List<String> fileIds: List of ids of files & folders

@param String objectFileName: Full object name of the file, e.g. 'cg__AccountFile__c'

@param Datetime expirationDate: Expire date for the link

@param String password: The minimum password length should be 4 characters

@param String IpRange: Two IP addresses as string separated by a “-“ dash

Z. createPublicAccessLink()

You can use SDriveTools API class to create a public access link, after configuring S-URL from S-Drive Configuration tab. To create a public access link using API class call the following method:

```
cg.SDriveTools.createPublicAccessLink(fileIds, objectFileName,
objectId, objectName, objectRelationshipName, isUpdatable,
isDeletable, isCreatable, expirationDate, password, IpRange);
```

The method signature is explained as follows:

@param List<String> fileIds: List of ids of files & folders

@param String objectFileName: Full object name of the file, e.g. 'cg__AccountFile__c'

@param String objectId: Id of parent object, i.e. for account files it is id of Account record

@param String objectName: Name of the parent object, i.e. for account files it is "Account"

@param String objectRelationshipName: The name of the relationship field that between parent and file objects. e.g. relationship field name of between account & account files is “Account__r”

@param Boolean isUpdatable: It enables/disables file edit feature for public user

@param Boolean isDeletable: It enables/disables file delete feature for public user

@param Boolean isCreatable: It enables/disables file upload & create new folder feature for public user

@param Datetime expirationDate: Expire date for the link

@param String password: The minimum password length should be 4 characters

@param String IpRange: Two IP addresses as string separated by a “-“ dash

AA. createPublicLink ()

You can use SDriveTools API class to create a public access link, after configuring S-URL from S-Drive Configuration tab. To create a public access link using API class call the following method:

```
createPublicLink(List<String> fileIds, String objectFileName, String
objectId, String objectName, String objectRelationshipName, Boolean
isAccessible, Boolean isUpdatable, Boolean isDeletable, Boolean
isCreatable, Datetime expirationDate, String password, String IpRange,
Boolean isDownloadLink, Boolean isOpen)
```

The method signature is explained as follows:

@param List<String> fileIds: List of ids of files & folders

@param String objectFileName: Full object name of the file, e.g. 'cg__AccountFile__c'

@param String objectId: Id of parent object, i.e. for account files it is id of Account record

@param String objectName: Name of the parent object, i.e. for account files it is "Account"

@param String objectRelationshipName: The name of the relationship field that between parent and file objects. e.g. relationship field name of between account & account files is “Account__r”

@param Boolean isUpdatable: It enables/disables file edit feature for public user

@param Boolean isDeletable: It enables/disables file delete feature for public user

@param Boolean isCreatable: It enables/disables file upload & create new folder feature for public user

@param Datetime expirationDate: Expire date for the link

@param String password: The minimum password length should be 4 characters

@param String IpRange: Two IP addresses as string separated by a “-“ dash

@param Boolean isDownloadLink: If true, file is downloaded

@param Boolean isOpen: If true, file is opened in component

II. S-Drive REST API

A. Introduction

The S-Drive REST services is an interface deployed within the S-Drive package available on the Salesforce App Exchange. It leverages Salesforce's servers and REST architecture. To have as many cases covered in interfacing with S-Drive we highly suggest developers to use the Force.com REST API developed by Salesforce found here.

B. Service Endpoint

HTTP web service methods deployed on a Salesforce org follows a pattern. It starts with the instance URL followed by the resource location. The format goes as

https://{*instance*}. salesforce.com/services/apexrest/cg/SDrive/{*resourcelocation*}

The instance is a variable unique for each org, it is the characters between "https://" and "salesforce.com" in your org URL. The resource location variable for S-Drive files is "files".

C. Authentication

Authentication for REST services deployed on Salesforce are done with OAuth 2.0 You may find Salesforce's documentation on using OAuth 2.0 for Salesforce authentication here.

To establish an authenticated connection to an outside app you would have to first create a "connected app" on Salesforce and get the Client Id and the Client Secret token. To find Salesforce's detailed documentation on connected apps you may go here.

Authentication is always included in the header.

D. Files Resource

Endpoint

https://{*instance*}.salesforce.com/services/apexrest/cg/SDrive/files

Methods

- GET
- POST
- PUT
- DELETE

Method Details

- **GET**

  Request Query Parameters

  fileObjectId: ids of the file you want to query

  parentId : ids of the parent object of the requested file. If file object type is an S3 object members of this array can be null or invalid input but array lengths must still match.

  timeValue : Expiration time of the links in seconds

  requestParameters (Optional) : Any remaining parameters will be included as additional query parameters for the Amazon file link(e.g. ('response-content-disposition', 'inline; filename=myfile.png')). For more information, visit the Amazon documentation:

  http://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectGET.html

  Response Body Parameters

  fileUrls : JSON array of requested file URLS.

  Type: string[]

  Status Codes
  200 - Object received
  403 - Wrong input formatting
  404 - False ids
  422 - Wrong input semantics

- **POST**

  Request Body Parameters

  objectId : Id of the parent object this file will be uploaded to

Type: string

multipartUpload : Do multipart Upload, used for big files(currently unavailable)

Type : Boolean

fileType : File Object the new files will be uploaded as.

Type: string

fileProperties : JSON array of file metadata of the uploaded files.

Type: fileProperty[]

policyMap : Key value pairs that represent the additional policy parameters used during upload

Type: string[]

Response Body Parameters

uploadRequestInfo : JSON object that details how to upload a file to Amazon S3

Type: uploadRequestInfo[]

Status Codes
201 - Object properly created
403 - Wrong input formatting
404 - False objectId
406 - Invalid input
412 - Missing input fields
422 - Wrong input semantics

Usage

Use POST to initiate an upload. Creates File Object records on S-Drive. Follow it by uploading the files using the data in uplodRequestInfo to Amazon S3 then finish with the PUT method

- **PUT**

Request Body Parameters

wipIds : JSON array of Salesforce ids that have completed their uploads to Amazon S3.

Type: string[]

Response Body Parameters

resultObjects : JSON object that details the status of each file's upload request.

Type: resultObject[]

Status Codes
200 - Upload complete
403 - Wrong input formatting

404 - False wipIds
422 - Wrong input semantics

Usage

Done after the POST method and uploading files to the correct location in Amazon S3.

- **DELETE**

Request Query Parameters

objectId : Id of the parent object.

wipId : Salesforce id of the object that will be deleted from S-Drive and Amazon S3.

Response Body Parameters

resultObjects : JSON object that details the status of each file's delete request.

Type: ResultObject[]

Status Codes
200 - Object deleted
403 - Wrong input formatting
404 - False objectIds
422 - Wrong input semantics

Usage

Deletes the File from Amazon S3 then removes the file record from Salesforce

E. File Objects Resource

Endpoint

https://{*instance*}. salesforce.com/services/apexrest/cg/SDrive/fileObjects

Methods

GET

Method Details

GET

Request Body Parameters

none

Response Body Parameters

fileObjects : List of Apex API names of SDrive file objects in the org.

Type: string[]

Status Codes
200 – Success

JSON Object Types

JSON Objects and their key value pairs that are used as arrays in our REST request and response bodys.

policyMap : Map of policy conditions and theirs values that represent the additional policy parameters used during upload (e.g. (*'$Content-Disposition'*, *'attachment; filename*)). For more information, visit the Amazon documentation:

http://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectPOST.html

FileProperty :
```
fileName : string
fileLocation : string
description : string
relationshipFieldName : string
fileType : string
fileSizeInBytes : string
```

UploadRequestInfo :
```
fileName : string
fileLocation : string
fileSize : string
fileType : string
wipFileId : string
signature : string
```

`policy : string`

ResultObject
`status : string` (success or fail)
`errormessage : string` (empty if status is success)
`wipField : string` (to see which file failed)

III. Automatic Folder Creation

If you want to build an automatic folder creation process in S-Drive for any type of object you have when you insert a new record, please refer to this section. We will use Account and cg__AccountFile__c object for our example.

1. For the object you have, create an Apex Trigger.
2. If you want this folder creation to take place after you create a new record, the trigger should work with an “after insert” statement.
3. In the Apex Trigger, paste the following lines:

```
trigger AccountFoldersTrigger on Account (after insert) {
      List<cg__AccountFile__c> folders = new List<cg__AccountFile__c>();
      for(Account c : Trigger.new{
            cg__AccountFile__c folder = new cg__AccountFile__c();
            folder.cg__WIP__c = false;
            folder.cg__Content_Type__c = 'Folder';
            folder.cg__File_Size_in_Bytes__c = 0;
            folder.cg__File_Name__c = 'Test Folder 1';
            folder.cg__Description__c = 'Some description for folder'; //use
a description field if needed.
            folder.cg__Account__c = c.Id;
            folders.add(folder);

            cg__AccountFile__c folder2 = new cg__AccountFile__c();
            folder2.cg__WIP__c = false;
            folder2.cg__Content_Type__c = 'Folder';
            folder2.cg__File_Size_in_Bytes__c = 0;
            folder2.cg__File_Name__c = 'Test Folder 2'; //use name as
appropriate
      folder2.cg__Description__c = 'Some description for folder'; //use a
description field if needed.
```

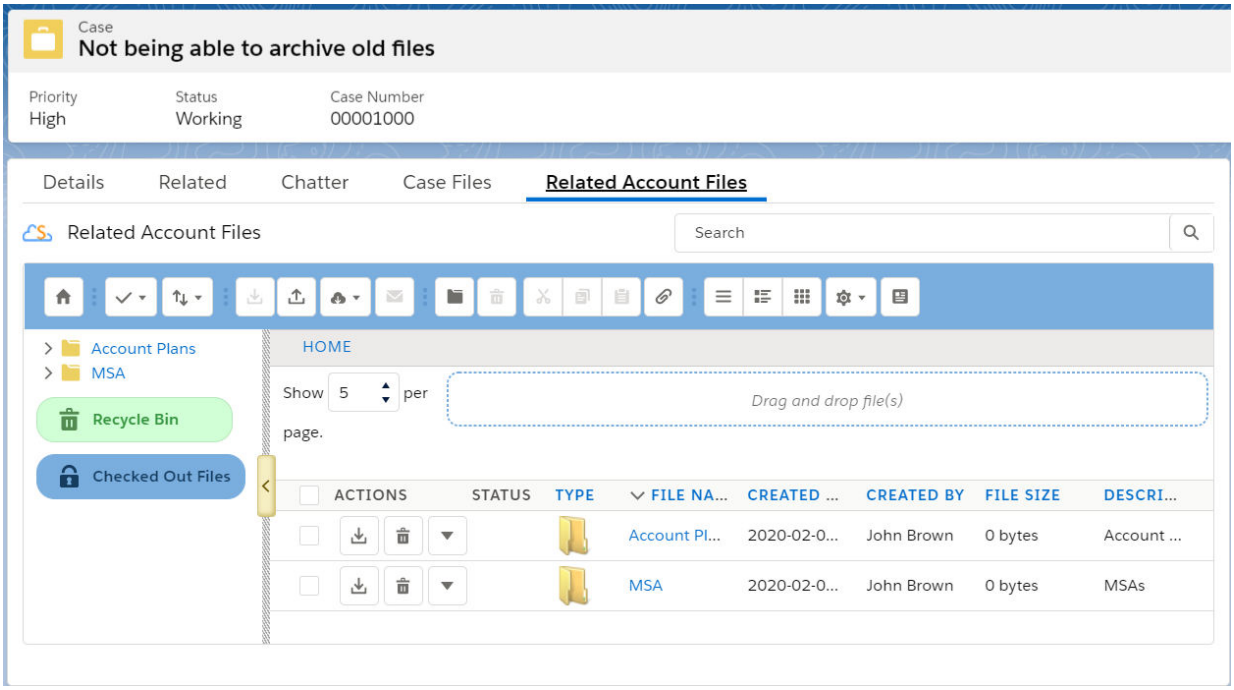
```
            folder2.cg__Account__c = c.Id;
            folders.add(folder2);

            if(folders.size() > 0){
                  insert folders;
            }
}
```

4. Add up to your requirement the number of folders created.
5. Change the Folder Names by changing the “File Name” section.
   For our example, the field is written as “cg__File_Name__c”.
6. You can add descriptions to your folders by changing the “Description” field.
   For our example, the field is written as “cg__Description__c”.
7. Make sure your Apex Trigger is active.
8. Test by inserting a new test record and you can use these folders according to your needs.

IV. Showing Related Files on a File Object

You can show parent object files on child objects and vice-versa but configuring multiple S-Drive Lightning components on the same record page. This requires creating a new Lightning Component and a new Apex Class is required. The Lightning component will have the S-Drive component in it with attributes set according to the related object file.

In this example, we will be showing how you can enable showing the related Account’s files in a Case record page in lightning.



1. The Lightning Component > RelatedAccountFiles.cmp
2. The Lightning Component JS Controller > RelatedAccountFilesController.js
3. The Apex Class > RelatedAccountFilesLightningCmpCtrl.apxc

The Lightning Component > RelatedAccountFiles.cmp

```
<aura:component controller="RelatedAccountFilesLightningCmpCtrl"
implements="flexipage:availableForRecordHome,force:hasRecordId" access="global" >
   <aura:attribute name="relatedAccountId" type="String"/>
   <aura:handler name="init" action="{!c.doInit}" value="{!this}"/>
     <aura:if isTrue="{!v.relatedAccountId}">
      <cg:SDrive title="Related Account Files"
                customObjectName="Account"
                customObjectFileName="AccountFile__c"
                fileNamespacePrefix="cg__"
                relationshipName="Account__r"
                componentId="AccountFilesCmp"
                recordId="{!v.relatedAccountId}"
                />
```

```
  </aura:if>
</aura:component>
```

The <cg:SDrive> component will have the following attributes according to your use case:

customObjectName: The object that you want to use S-Drive with

customObjectFileName: The object you created for recording your S-Drive files

fileNamespacePrefix: For file objects coming with S-Drive package this attribute should be “cg__”, for other objects you have in your organization it will be empty as “”

relationshipName: The relationship name between the parent object and the file object

componentId: This componentId is advised to be set as unique when you have multiple components in one record page

recordId: This will be set as the related parent object’s id that you want to show the files of

If you want to show a specific record’s files in every record’s page, you can write the ID of the object directly in to recordId attribute. If not, you will need to create the Javascript and Apex controllers.

The Lightning Component JS Controller > RelatedAccountFilesController.js

```
({
        doInit : function(component, event, helper) {
    var caseId = component.get("v.recordId");
    var action = component.get("c.getRelatedAccountId");
    action.setParams({
      caseId : caseId
    });
    action.setCallback(this, function(response) {
      var r = response.getReturnValue();
      var state = response.getState();
```

```
        if (component.isValid() && state === "SUCCESS"){
            component.set("v.relatedAccountId", r);
        }
    });
    $A.enqueueAction(action);
        }
})
```

The Apex Class > RelatedAccountFilesLightningCmpCtrl.apxc

```
public class RelatedAccountFilesLightningCmpCtrl {
   @AuraEnabled
   public static String getRelatedAccountId(String caseId){
      String queryStr = 'SELECT ID, AccountId FROM Case WHERE Id = :caseId LIMIT 1';
      List<Case> caseList = Database.query(queryStr);
      String relatedAccountId = '';

      if(null != caseList && caseList.size() > 0){
         relatedAccountId = caseList[0].AccountId;
      }
      return relatedAccountId;
   }
}
```

V. Branding Public Share Site

To style and use your branding colors in the S-URL site – (The visualforce page that is the Active Site Home Page – cg.SURLRedirect.page), you can follow these steps.

1. Create a CSS stylesheet in .css format and upload it as a static resource to Salesforce. You can name it as "sdrivecustom.css". You can upload a static resource from Setup > Static Resources > New.

   The stylesheet should be in the following format.

```
1  body {
     background: url('https://www.cyangate.com/wp-content/uploads/2015/02/bgSlider.jpg') !important;
   }

2  #SDrive {
     position: absolute !important;
     width: calc(100% - 38px) !important;
     bottom: 0 !important;
   }

3  #SDrive .slogan {
     color: #192B55 !important;
   }
```

4

```
.sdrive-btn {
    border-radius: 5px !important;
    background-color: #192B55 !important;
    border: 1px solid #314472 !important;
}
```

5

```
.sdrive-btn:hover {
    background-color: #314472 !important;
}
```

6

```
.cgToolbar .slds-button{
    background-color: #64C8CB !important;
    color: white !important;
    border: 1px solid #64C8CB !important;
}
```

7

```
.slds-scope .slds-button_icon-border-filled[disabled], .slds-scope .slds-button_icon-border-filled:disabled{
    background-color: #96c5c7 !important;
}
```

8

```
#SDriveCmp{
    background-color: #a0a0a0 !important;
    border: 1px solid #314472 !important;
```

```
}

9  #SDriveCmp .slds-box{
       border: 1px solid #314472 !important;
   }

10 #global-search-01{
       background: #D3D3D3;
       border: 2px solid #314472;
   }

11 .cgSDrive .sd-search-btn.slds-input__icon{
       border-radius: 2px !important;
       width: 34px !important;
       top: 2px !important;
       height: 29px !important;
       color: #314472 !important;
   }

12 .file-list-table{
       background-color: #D3D3D3 !important;
   }
13 .slds-scope .slds-table:not(.slds-no-row-hover) tbody tr:hover > td{
       background-color:#e0e0e0 !important;
   }
```

```
14  .slds-scope .slds-button_icon-border-filled{
        background-color: #64C8CB !important;
    }

15  .community-toolbar{
        background-color: #192B55 !important;
    }

16  .cgActions lightning-primitive-icon{
        color:#fff;
    }

17  .pass-container{
        background-color: #192B55 !important;
        color: white !important;
        border: 1px solid #64C8CB !important;
    }

18  .pass-container input[type='password']{
        background-color: #D3D3D3 !important;
        border-radius: 2px !important;
    }

19  .pass-container input[type='submit']{
```

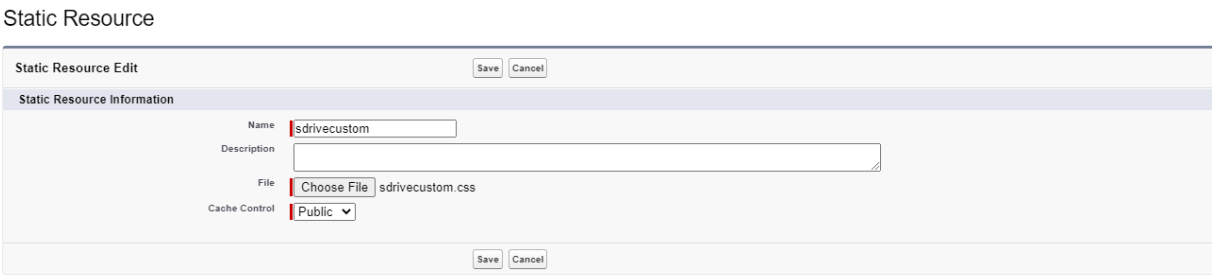
```
    background: #64C8CB !important;

    border: 1px solid #959595 !important;

    font-size: 1em !important;

}
```

Note: It is required that you set your stylings as "!important", in order to override the styles of regular S-URL page.

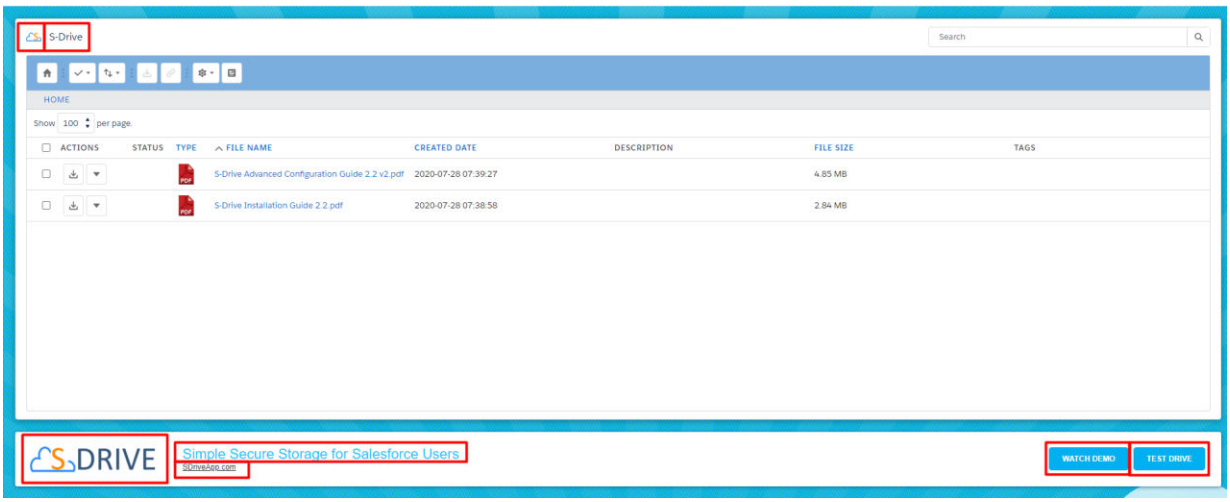
Use the following styles for the following scenarios to change:

- The background color or image of the page → 1
- The width or position of the card or hide the card at the bottom of the page → 2
- The colors of the slogan → 3
- The button styles and button hover colors → 4-5
- The S-Drive Toolbar buttons → 6
- The S-Drive component's colors → 7
- The S-Drive component's background color → 8
- The search-bar colors and style →10-11
- The background color of the S-Drive file-list → 12
- The S-Drive toolbar color → 15
- The colors of action menu items and quick actions → 16
- The password pop-up style → 17-18-19

You can set the Cache Control as **public** when you upload the static resource.



2. Create a Javascript script in .js format and upload it as a static resource to format the following sections in the S-URL page. You can name it as "sdrivecustomscript.js". You can upload a static resource from Setup > Static Resources > New.




```
window.addEventListener('load', function () {
    setSDriveComponentTitle();
    setDriveComponentBrandLogo();
    setSurlPageSlogan();
    setSurlPageSloganUrl();
    setSurlPageButton1();
    setSurlPageButton2();
    setSurlPageBrandLogo();
    setSurlPageBrandLogoUrl();
});

function setSDriveComponentTitle(){
    var element = document.querySelector("span[title='S-Drive ']");
    if(!element){
        setTimeout(function(){
            setSDriveComponentTitle();
        },200);
```

```
    }
    else{
        element.innerHTML = 'CyanGate S-Drive';
    }
}

function setDriveComponentBrandLogo(){
    var element = document.querySelector("img[title='S-Drive']");
    if(!element){
        setTimeout(function(){
            setDriveComponentBrandLogo();
        },200);
    }
    else{
        element.setAttribute('src', 'https://www.cyangate.com/wp-
content/uploads/2019/10/cyangate_logo.png');
    }
}

function setSurlPageSlogan(){
    var element = document.querySelector("span[class='slogan']");
    if(!element){
        setTimeout(function(){
            setSurlPageSlogan();
        },200);
    }
```

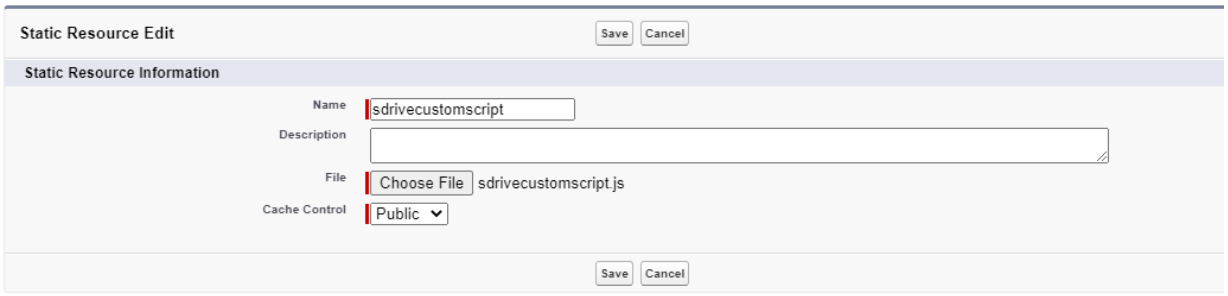
```
    else{
        element.innerHTML = 'CyanGate Storage';
    }
}

function setSurlPageSloganUrl(){
    var element = document.querySelector("a[id='slogan-url']");
    if(!element){
        setTimeout(function(){
            setSurlPageSloganUrl();
        },200);
    }
    else{
        element.innerHTML = 'CyanGate.com';
        element.setAttribute('href', 'https://www.cyangate.com/');
    }
}

function setSurlPageButton1(){
    var element = document.querySelector("a[id='surl-button-1']");
    if(!element){
        setTimeout(function(){
            setSurlPageButton1();
        },200);
    }
```

```
    else{
        element.innerHTML = 'BUTTON 1';
        element.setAttribute('href', 'https://www.cyangate.com/');
    }
}

function setSurlPageButton2(){
    var element = document.querySelector("a[id='surl-button-2']");
    if(!element){
        setTimeout(function(){
            setSurlPageButton2();
        },200);
    }
    else{
        element.innerHTML = 'BUTTON 2';
        element.setAttribute('href', 'https://www.sdriveapp.com/');
    }
}

function setSurlPageBrandLogo(){
    var element = document.querySelector("img[id='surl-brand-logo']");
    if(!element){
        setTimeout(function(){
            setSurlPageBrandLogo();
        },200);
```

```
    }
    else{
        element.setAttribute('src', 'https://www.cyangate.com/wp-
content/uploads/2019/10/cyangate_logo.png');
    }
}


function setSurlPageBrandLogoUrl(){
    var element = document.querySelector("a[id='surl-brand-url']");
    if(!element){
        setTimeout(function(){
            setSurlPageBrandLogoUrl();
        },200);
    }
    else{
        element.setAttribute('href', 'https://www.cyangate.com/');
    }
}
```

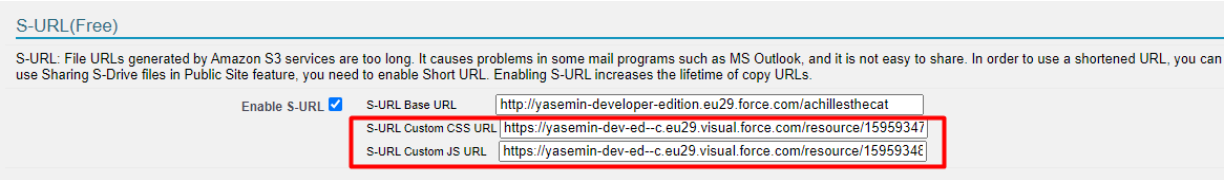
You can set the Cache Control as **public** when you upload the static resource.

Static Resource

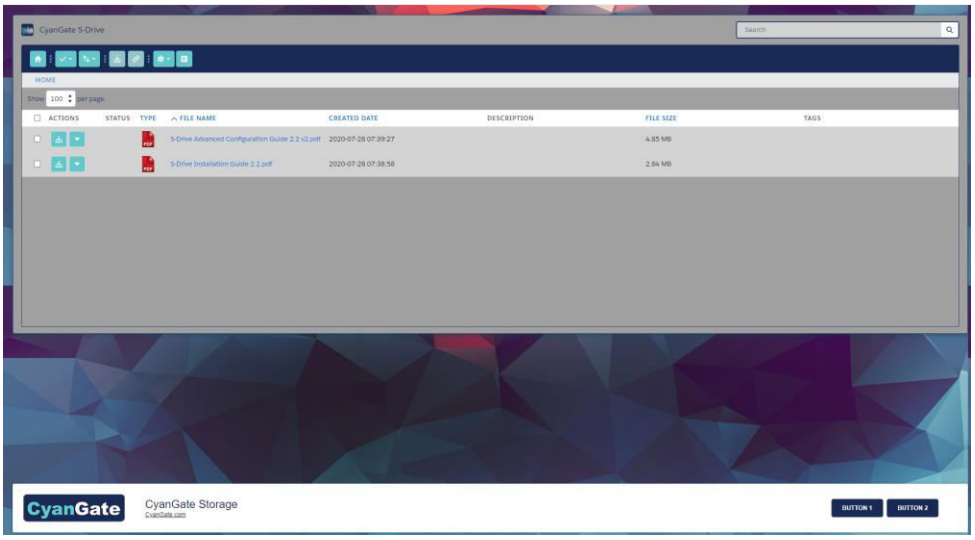


3. Click on “View File” link in the Static resources page and copy the URLs for both resources from the browser.

4. Navigate to S-Drive Configuration page and click on “Micro Services” tab to paste the URLs for the custom style sheet and the custom script that you installed as a static resource, respectively.



5. The S-URL page will look like the following screenshot when the example css and js changes are applied.



VI. S-DRIVE SUPPORT

You can contact S-Drive Support team for any questions or problems that you couldn't solve using S-Drive documents:

1. Open a Ticket at Support Site: sdriveapp.com/support
2. Send an Email: sdrive@sdriveapp.com

You can find up-to-date product information, documents, tutorial videos, tools in our web page: www.sdriveapp.com