Smart S3 Connector For Creatio

Installation

How to use

To perform the installation, use the latest version of the application from the marketplace. The application provides basic functionality for integrating with S3 storage.

To begin using the integration with S3 storage provided by the application, change the following settings:

- For system settings with code **SmartS3ConnectionString**, update connection string that contains proper values for integration. The buckets should be created in advance. Example: *ServiceUrl=https://s3.host; AccessKey= accesskey; SecretKey= secretkey; ObjectBucketName=object-bucket; RecycleBucketName=recycle-bucket;*
- For system settings with code **ActiveFileContentStorage**, change value to **"S3 storage (SmartS3Connector)"**.

After completing the installation of the application, compile the package.

Customization

Application hierarchy

Before making any customizations, you need to create your own composable application with a dependency on Smart S3 Connector For Creatio. Go to Settings -> Application Hub and create a new application based on the Custom template. Then, go to package properties and add a new dependency from the CrtSmartS3ConnectorApp package.

Crete and register a new provider with basic behavior

Implement your own S3 provider in code. Open <PACKAGE_NAME>.csproj file in IDE and add a new C# class inherited from SmartS3FileContentStorage.

```
public class PeriodicalS3FileContentStorage : SmartS3FileContentStorage
{
    public PeriodicalS3FileContentStorage(
            SmartS3SettingsProvider settingsProvider,
            SmartS3UploadBufferManager bufferManager,
            SmartS3DataStoreMultipartUploadInfoStorage multipartUploadInfoStorage)
        : base(
            settingsProvider,
            bufferManager,
            multipartUploadInfoStorage) {
    }
}
```

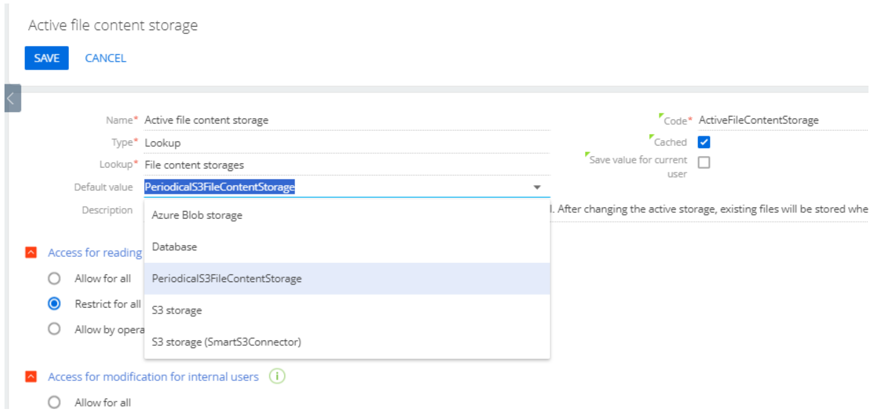
To register a new provider in the system and start working with it, add a new record in the lookup "File Content Storages" (SysFileContentStorage entity). That record should have the following values in its fields:

- Name (Code) – Unique name based on application name
- Code (Code) – Unique core based on application name.
- Storage type name (TypeName) – A string in format {NAMESPACE.CLASS}, {ASSEMBLY} of new provider

Examples for the fields mentioned above: "PeriodicalS3FileContentStorage", "UsrPeriodicalS3Connect.PeriodicalS3FileContentStorage, UsrPeriodicalS3Connect", "PeriodicalS3FileContentStorage"

Don't forget to bind this record's data in the package to provide functionality deliverables.

Switch the active file content storage to your new provider by changing the system settings value with the code ActiveFileContentStorage.



Active file content storage

SAVE CANCEL

Name*	Active file content storage	Code*	ActiveFileContentStorage
Type*	Lookup	Cached	✓
Lookup*	File content storages	Save value for current user	☐
Default value	PeriodicalS3FileContentStorage		
Description	l. After changing the active storage, existing files will be stored whe		

Azure Blob storage
Database
PeriodicalS3FileContentStorage
S3 storage
S3 storage (SmartS3Connector)

Access for reading
Allow for all
[✓ Restrict for all]
Allow by opera

Access for modification for internal users
Allow for all

Override bucket naming strategy

By default, bucket names are read from the connection string in system settings. By overriding the GetObjectBucketName and GetRecycleBucketName methods in your provider, you can define a flexible approach for generating bucket names.

In the example below, any files will be put in a bucket that matches the year of file creation on the system.

Bucket for deleted objects remains unchanged.

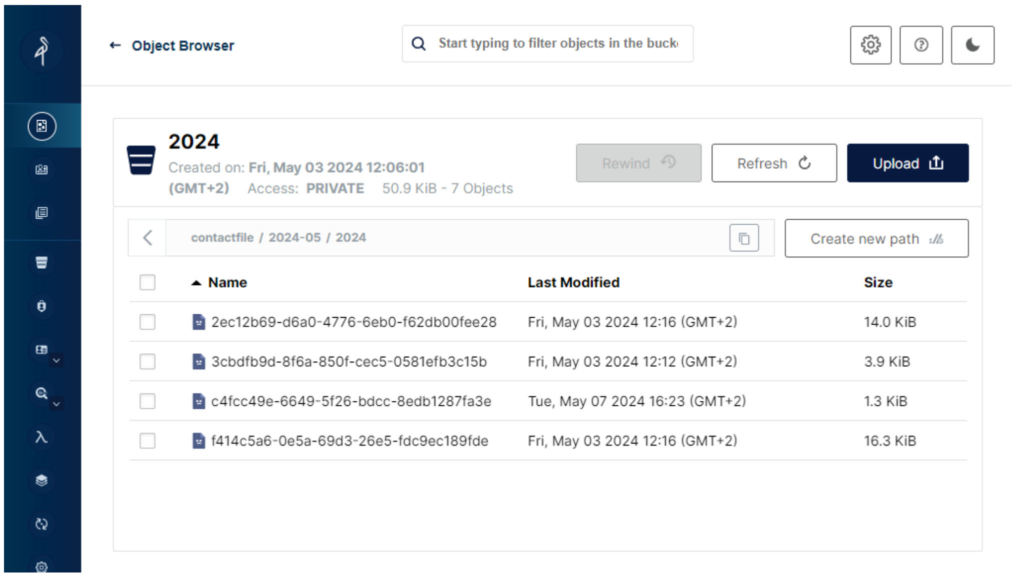
```
public class PeriodicalS3FileContentStorage : SmartS3FileContentStorage
{
   ...
    protected override string GetObjectBucketName(FileMetadata fileMetaData) {
        if (!(fileMetaData is EntityFileMetadata entityFileMetadata)) {
            throw new NotSupportedException();
        }
        var createdOnYear =  entityFileMetadata.CreatedOn.ToString("yyyy");
        return createdOnYear;
    }

    protected override string GetRecycleBucketName(FileMetadata fileMetaData) {
        return base.GetRecycleBucketName(fileMetaData);
    }

}
```

Please, pay attention that backets should be created in advance.

Override object path generation

By default, an object has a path consisting of the file schema name and record ID. By overriding the GetObjectKey method, you can use another approach to forming the path. The example below demonstrates generating a path based on the created-on value.

```
public class PeriodicalS3FileContentStorage : SmartS3FileContentStorage
{
    ...
    protected override string GetObjectKey(FileMetadata fileMetadata) {
        if (!(fileMetadata is EntityFileMetadata entityFileMetadata)) {
            throw new NotSupportedException();
        }
        string entitySchemaName = entityFileMetadata.EntitySchemaName.ToLowerInvariant();
        string recordId = entityFileMetadata.RecordId.ToString();
        var createdOnYear = entityFileMetadata.CreatedOn.ToString("yyyy");
        var monthOnYear = entityFileMetadata.CreatedOn.ToString("MM");
        return $"{entitySchemaName}/{createdOnYear}-{monthOnYear}/{recordId}";
    }
}
```



2024
Created on: Fri, May 03 2024 12:06:01 (GMT+2) Access: PRIVATE 50.9 KiB - 7 Objects

contactfile / 2024-05 / 2024

Name	Last Modified	Size
2ec12b69-d6a0-4776-6eb0-f62db00fee28	Fri, May 03 2024 12:16 (GMT+2)	14.0 KiB
3cbdfb9d-8f6a-850f-cec5-0581efb3c15b	Fri, May 03 2024 12:12 (GMT+2)	3.9 KiB
c4fcc49e-6649-5f26-bdcc-8edb1287fa3e	Tue, May 07 2024 16:23 (GMT+2)	1.3 KiB
f414c5a6-0e5a-69d3-26e5-fdc9ec189fde	Fri, May 03 2024 12:16 (GMT+2)	16.3 KiB

Override connection parameters

In exceptional cases, you may need to override the behavior of the provider settings that return the settings for the connection to S3. Create your own class, specifying the SmartS3SettingsProvider class as its parent.

```
public class PeriodicalS3SettingsProvider : SmartS3SettingsProvider
{
    public PeriodicalS3SettingsProvider(UserConnection userConnection)
        : base(userConnection) {
    }
}
```

Then, you need to start using settings provider in the constructor.

```
public class PeriodicalS3FileContentStorage : SmartS3FileContentStorage
{
    public PeriodicalS3FileContentStorage(
            PeriodicalS3SettingsProvider settingsProvider,
            SmartS3UploadBufferManager bufferManager,
            SmartS3DataStoreMultipartUploadInfoStorage multipartUploadInfoStorage)
        : base(
            settingsProvider,
            bufferManager,
            multipartUploadInfoStorage) {
    }
}
```

By overriding the GetConnectionStringSysSettingCode method, you can set the code of a system setting where the connection string to S3 is stored. In most cases, storing connection settings in separate system settings will be sufficient.

By overriding the GetS3Settings method, you can build and return an object containing connection parameters.

```
public class PeriodicalS3SettingsProvider : SmartS3SettingsProvider
{
    ...
    protected override string GetConnectionStringSysSettingCode() {
        return base.GetConnectionStringSysSettingCode();
    }

    public override SmartS3Settings GetS3Settings() {
        return base.GetS3Settings();
    }
}
```