Creatio Academy

Back-end development

API for file management

Version 8.0

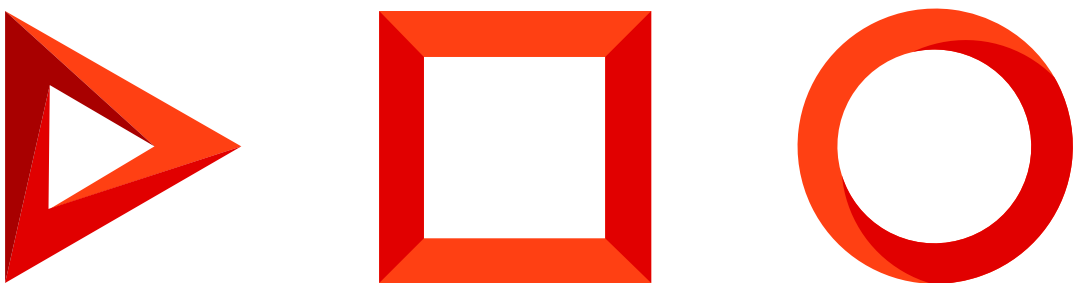

Table of Contents

API for file management

Medium

Creatio core provides a set of file management classes and interfaces in version 7.17.2 and up:

- `Terrasoft.File.Abstractions` namespace – interfaces and abstract classes that specify the file management logic in Creatio.
- `Terrasoft.File` namespace – concrete implementations of the abstractions used in Creatio.

The location of a file and file locators

The location of a file is specified using a **file locator**, which is an object that implements the `Terrasoft.File.Abstractions.IFileLocator` interface.

A file locator must contain a **unique file identifier**: `RecordId`.

You can create different file locator implementations for different file storages. Depending on the file storage mechanics, the file locator may have extra properties for the identification of the file location. For example, the `Terrasoft.File.EntityFileLocator` class is an implementation of the file locator for the current [Attachments] file storage in Creatio. An `EntityFileLocator` object has the base `RecordId` property and an `EntitySchemaName` property: the name of the object schema where the file is stored, for example: "ActivityFile" or "CaseFile".

All file management methods employ file locators.

Files and file storages

File structure in Creatio:

- file metadata
- file content.

Metadata describe the file properties:

- name
- size in bytes
- creation date, etc.

File metadata are based on the `Terrasoft.File.Abstractions.Metadata.FileMetadata` abstract class. Let's consider the `Terrasoft.File.Metadata.EntityFileMetadata` class as its concrete implementation. This class describes the metadata of the [Attachments] object files.

"Content" is the contents of the file.

In Creatio, the file metadata and content are stored in different **storages**.

- The file metadata storage must implement the `Terrasoft.File.Abstractions.Metadata.IFileMetadataStorage` interface.

- The file content storage must implement the `Terrasoft.File.Abstractions.Metadata.IFileContentStorage` interface.

Specific implementations of these interfaces cover all subtle aspects of interaction with different file storage systems: Creatio [Attachments], the filesystem of the server, Amazon S3. Google Drive, etc.

The `Terrasoft.File.Abstractions.IFile` interface provides essential file management methods applicable in any file storage. In terms of Creatio, a **file** is an implementation of this interface. The methods of this interface are used for asynchronous file management. Synchronous file management methods are available in the `Terrasoft.File.Abstractions.FileUtils` derived class.

Get a file (`IFileFactory`)

The `Terrasoft.File.Abstractions.IFileFactory` interface provides methods for creating and getting objects of a class that implements the `Terrasoft.File.Abstractions.IFile` interface. This interface is implemented by a factory accessed using the methods of the `Terrasoft.File.FileFactoryUtils` class that extends the `UserConnection` class. Therefore, an instance of `UserConnection` or `SystemUserConnection` is required for file management.

The location of a new or existing file is unambiguously determined by its file locator. To access an existing file, its `RecordId` identifier is required. For a new file, create an identifier and pass it to the locator creation method.

Implement and register a new file storage type

To implement a new file storage type:

1. Create an implementation of the `Terrasoft.File.Abstractions.Content.IFileContentStorage` interface that describes the API required for working with the file storage.
2. If the current file metadata storage (`Terrasoft.File.Metadata.EntityFileMetadataStorage`) is not sufficient for your tasks, implement your own metadata storage, metadata type, and file locator type:
   - The file metadata storage must implement the `Terrasoft.File.Abstractions.Metadata.IFileMetadataStorage` interface.
   - The file locator must implement the `Terrasoft.File.Abstractions.IFileLocator` interface.
   - The metadata class must inherit from the `Terrasoft.File.Abstractions.Metadata.FileMetadata` class.
3. The new file content storage and file metadata storage must be registered in the corresponding “SysFileContentStorage” and “SysFileMetadataStorage” lookups.

File management exceptions

Exception type	Message	Exception conditions
`Terrasoft.File.Abstractions.FileNotFoundByLocatorException`	File not found by locator '{locator_type} {locator.ToString}'	When accessing any properties or methods of the `IFile` interface if the file metadata are not found.
`System.InvalidOperationException`	Can't delete new file: '{locator_type} {locator.ToString}'	When attempting to delete a newly created file: `FileMetadata.StoringState == FileStoringState.New`
`Terrasoft.Common.NullOrEmptyException`	File name cannot be null or empty	When attempting to save a file with the `Name` field left blank
`System.InvalidOperationException`	Can't find a metadata storage for the '{locator_type}' locator type	If a file metadata storage compatible by the locator type is not found.
`System.InvalidOperationException`	Can't find a content storage for the '{metadata_type}' metadata type	If a file content storage compatible by the metadata type is not found.

File management examples

Medium

Example. Get an instance of a class that implements the IFile interface.

Get IFile for an existing file (option 1)

```
/* Get the file factory. */
IFileFactory fileFactory = UserConnection.GetFileFactory();
/* Creating a file locator for a file identified by the recordId and stored in the "ActivityFile
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
/* Pass the created locator to the Get method of the factory. */
IFile file = fileFactory.Get(fileLocator);
/* This will return an instance of a class that implements the IFile interface. Use it to manipu
```

Get IFile for an existing file (option 2)

```
/* Create a file locator for a file identified by the recordId and stored in the "ActivityFile"
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
/* Pass the created locator to the GetFile extending method. */
IFile file = UserConnection.GetFile(fileLocator);
/* This will return  an instance of a class that implements the IFile interface. Use it to manip
```

Get IFile for a new file (option 1)

```
/* Get the file factory. */
IFileFactory fileFactory = UserConnection.GetFileFactory();
/* Creating a unique identifier for the new file. */
Guid recordId = Guid.NewGuid();
/* Create a file locator for a new file identified by the recordId and stored in the "ActivityFi
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
/* Pass the created locator to the Create method of the factory. */
IFile file = fileFactory.Create(fileLocator);
/* This will return an instance of a class that implements the IFile interface. Use it to manipu
```

Get IFile for a new file (option 2)

```
/* Create a unique ID for the new file. */
Guid recordId = Guid.NewGuid();
/* Create a file locator for a new file identified by the recordId and stored in the "ActivityFi
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
/* Pass the created locator to the CreateFile extension method. */
IFile file = UserConnection.CreateFile(fileLocator);
/* This will return an instance of a class that implements the IFile interface. Use it to manipu
```

Example. Create a new file bound to an [*Activities*]([*Activities*]) record.

Creating a new file

```
/* Create a unique ID for the new file. */
Guid fileId = Guid.NewGuid();
/* Create a file locator for the new file. */
var fileLocator= new EntityFileLocator("ActivityFile", fileId);
/* Get an IFile object for the new file. */
```

```
IFile file = UserConnection.CreateFile(fileLocator);
/* There is no file metadata or file content in the available file storages. Specify the file na
file.Name = "New file";
/* Set an attribute for the new file: bind the file to an Activity record with the activityId ke
file.SetAttribute("ActivityId", activityId);
/* Save the file metadata Do this BEFORE saving the content. */
file.Save();
/* byte[] content – this is the file contents. */
var content = new byte[] {0x12, 0x34, 0x56};
using (var stream = new MemoryStream(content)) {
    /* Save the content to the database. FileWriteOptions.SinglePart implies that the contents w
    file.Write(stream, FileWriteOptions.SinglePart);
}
```

Example. Get the file content.

Read the file content

```
using Terrasoft.Common;

var content = new byte[]();
Guid recordId = Guid.NewGuid();

/* Create a file locator for the new file. */
var fileLocator = new EntityFileLocator("ActivityFile", recordId);
IFile file = UserConnection.GetFile(fileLocator);

/* Read the contents of the file in the content byte array. Remember to free the stream object b
using (Stream stream = file.Read()) {
   content = stream.ReadToEnd();
}
```

Example. Copy a file, then move it to a new location and delete it.

Copy, move, and delete a file

```
var content = new byte[]();
/* Get the file with the file locator. */
var fileLocator = new EntityFileLocator("ActivityFile", fileId);
IFile file = UserConnection.GetFile(fileLocator);
/* Read the contents of the file in the content byte array. Remember to free the stream object b
using (Stream stream = file.Read()) {
```

```
    content = stream.ReadToEnd();
}
/* Copy the file. */

/* Create a new IFile to copy the current file. */
Guid copyFileId = Guid.NewGuid();
var copyFileLocator = new EntityFileLocator("ActivityFile", copyFileId);
IFile copyFile = UserConnection.CreateFile(copyFileLocator);
copyFile.Name = file.Name + " - Copy";
copyFile.Save();

/* Copy the contents of the first file to the new file. */
copyFile.Write(new MemoryStream(content), FileWriteOptions.SinglePart);

/* An alternate way to copy the file. */
file.Copy(copyFile);

/* Move the file. */

/* Create a new file to copy the current file. */
Guid moveFileId = Guid.NewGuid();
var moveFileLocator = new EntityFileLocator("ContactFile", moveFileId); IFile moveFile = UserCor
moveFile.Save();

/* Move to the new location. */
file.Move(moveFile);

/* Delete the original file. */
file.Delete();
```

Implement a file content storage

Medium

Example. Create a class that implements the IFileContentStorage interface to create a content storage in the file system.

Implement a file content storage

```
namespace Terrasoft.Configuration
{
   using System.IO;
   using System.Threading.Tasks;
   using Terrasoft.File.Abstractions;
   using Terrasoft.File.Abstractions.Content;
```

```
using Terrasoft.File.Abstractions.Metadata;
using Terrasoft.File.Metadata;

/// <summary>
/// Content storage in the file system.
/// </summary>
public class FsFileBlobStorage : IFileContentStorage
{

   // A root path to the storage.
   private const string BaseFsPath = "C:\\FsStore\\";

   private static string GetPath(FileMetadata fileMetadata) {
      var md = (EntityFileMetadata)fileMetadata;
      string key = $"{md.EntitySchemaName}\\{md.RecordId}_{fileMetadata.Name}";
      return Path.Combine(BaseFsPath, key);
   }

   public Task<Stream> ReadAsync(IFileContentReadContext context) {
      string filePath = GetPath(context.FileMetadata);
      Stream stream = System.IO.File.OpenRead(filePath);
      return Task.FromResult(stream);
   }

   public async Task WriteAsync(IFileContentWriteContext context) {
      string filePath = GetPath(context.FileMetadata);
      FileMode flags = context.WriteOptions != FileWriteOptions.SinglePart
         ? FileMode.Append
         : FileMode.OpenOrCreate;
      string dirPath = Path.GetDirectoryName(filePath);
      if (!Directory.Exists(dirPath)) {
         Directory.CreateDirectory(dirPath);
      }
      using (var fileStream = System.IO.File.Open(filePath, flags)) {
         context.Stream.CopyToAsync(fileStream);
      }
   }

   public Task DeleteAsync(IFileContentDeleteContext context) {
      string filePath = GetPath(context.FileMetadata);
      System.IO.File.Delete(filePath);
      return Task.CompletedTask;
   }

   public Task CopyAsync(IFileContentCopyMoveContext context) {
      string sourceFilePath = GetPath(context.SourceMetadata);
      string targetFilePath = GetPath(context.TargetMetadata);
      System.IO.File.Copy(sourceFilePath, targetFilePath);
      return Task.CompletedTask;
```

```
      }

      public Task MoveAsync(IFileContentCopyMoveContext context) {
         string sourceFilePath = GetPath(context.SourceMetadata);
         string targetFilePath = GetPath(context.TargetMetadata);
         System.IO.File.Move(sourceFilePath, targetFilePath);
         return Task.CompletedTask;
      }
   }
}
```

IFile interface C#

Medium

`Terrasoft.File.Abstractions` namespace.

The `Terrasoft.File.Abstractions.IFile` interface provides essential file management methods applicable in any file storage. The methods of this interface are used for asynchronous file management. Synchronous file management methods are available in the `Terrasoft.File.Abstractions.FileUtils` derived class.

Properties

`FileLocator` `IFileLocator`

The file locator connected to the current instance of the class that implements the `IFile` interface.

`Name` `string`

File name.

`Length` `long`

The size of the current file in bytes.

`CreatedOn` `DateTime`

Date and time the file was created.

`ModifiedOn` `DateTime`

Date and time the file was modified.

`Exists` `bool`

Checks if the current file exists.

Methods

`Task CopyAsync(IFile target)`

Copies the current file to the new `target` asynchronously.

`Task MoveAsync(IFile target)`

Moves the current file to the new `target` asynchronously.

`Task DeleteAsync()`

Deletes the current file asynchrously.

`Task WriteAsync(Stream stream, FileWriteOptions writeOptions)`

Writes the contents of the current file to the `stream` asynchronously.

`Task<Stream> ReadAsync()`

Reads the contents of the current file asynchronously.

`Task SaveAsync()`

Saves the metadata of the current file asynchronously.

`void SetAttribute<TValue>(string name, TValue value)`

Sets `value` of the `name` attribute for the current file.

`TValue GetAttribute<TValue>(string name, TValue defaultValue)`

Returns the attribute value or the default value for the current file.

IFileContentStorage interface C#

Medium

`Terrasoft.File.Abstractions` namespace.

The `Terrasoft.File.Abstractions.IFileContentStorage` interface provides the essential methods for file storage management.

Methods

`Task<Stream> ReadAsync(IFileContentReadContext context)`

Reads the file content.

`Task WriteAsync(IFileContentWriteContext context)`

Writes the file content.

`Task DeleteAsync(IFileContentDeleteContext context)`

Deletes the file content.

`Task CopyAsync(IFileContentCopyMoveContext context)`

Copies the file content.

`Task MoveAsync(IFileContentCopyMoveContext context)`

Moves the file content.

IFileFactory interface C#

Medium

`Terrasoft.File.Abstractions` namespace.

The `Terrasoft.File.Abstractions.IFileFactory` interface provides a set of methods to get or create an instance of the class that implements the `Terrasoft.File.Abstractions.IFile` interface.

Properties

`UseRights` `bool`

Determines if the permissions of the user are considered when creating a file.

Methods

`IFile Get(IFileLocator fileLocator, FileOptions options)`

Returns an instance of the class that implements the `IFile` interface with the specified `options` parameters from a given `fileLocator`.

`IFile Create(IFileLocator fileLocator, FileOptions options)`

Creates an instance of the class that implements the IFile interface with the specified `options` parameters for a given `fileLocator`.

EntityFileLocator class C#

Medium

`Terrasoft.File` namespace.

The class implements the `IFileLocator` interface for the current Creatio file storage.

Constructors

`EntityFileLocator()`

Creates a new `EntityFileLocator` instance.

`EntityFileLocator(string entitySchemaName, Guid recordId)`

Creates a new `EntityFileLocator` instance for the specified `recordId` file bound to the `entitySchemaName` object schema.

Properties

`EntitySchemaName string`

Metadata – the name of the object that stores the file.

`RecordId Guid`

Metadata – the file ID

EntityFileMetadata class C#

Medium

`Terrasoft.File` namespace.

The `Terrasoft.File.EntityFileMetadata` class implements the Terrasoft.File.Abstractions.Metadata.FileMetadata abstract class. This class describes the file metadata in the [Attachments] object and provide the methods to manage them.

Constructors

`EntityFileMetadata(EntityFileLocator fileLocator)`

Creates an `EntityFileMetadata` instance for a given `fileLocator`.

Properties

`Attributes IReadOnlyDictionary<string, object>`

Collection of attribute values.

`RecordId Guid`

File ID.

`EntitySchemaName string`

The name of the object that stores the file.

Methods

`override void SetAttribute<TValue>(string name, TValue value)`

Set the additional `name` file attribute to the specified `value`.

`override TValue GetAttribute<TValue>(string name, TValue defaultValue)`

Returns the specified value or the `defaultValue` of the additional `name` attribute .

FileFactoryUtils class C#

Medium

`Terrasoft.File` namespace.

The `Terrasoft.File.FileFactoryUtils` class provides extension methods for the `UserConnection` class and factory class that implements the `Terrasoft.File.AbstractionsIFileFactory` interface. This way, the class

provides access to the factory for creating new or getting existing files. Therefore, an instance of `UserConnection` or `SystemUserConnection` is required for file management.

Methods

`static IFileFactory GetFileFactory(this UserConnection source)`

An extension method for the `UserConnection` class that returns an instance of the class that implements the `IFileFactory` interface.

`static IFile GetFile(this UserConnection source, IFileLocator fileLocator)`

An extension method for the `UserConnection` class that returns an instance of the class that implements the `IFile` interface from a given `fileLocator`.

`static IFile CreateFile(this UserConnection source, IFileLocator fileLocator)`

An extension method for the `UserConnection` class that creates an instance of the class that implements the `IFile` interface from a given `fileLocator`.

`static IFile Get(this IFileFactory source, IFileLocator fileLocator)`

An extension method for the class that implements the `IFileFactory` interface. Returns an instance of the class that implements the `IFile` interface for a given `fileLocator`.

`static IFile Create(this IFileFactory source, IFileLocator fileLocator)`

An extension method for the class that implements the `IFileFactory` interface. Creates an instance of the class that implements the `IFile` interface for a given `fileLocator`.

`static IFileFactory WithRightsDisabled(this IFileFactory source)`

An extension method for the class that implements the `IFileFactory` interface. Returns an instance of the class that implements the `IFileFactory` interface configured without the access permissions of the user.

FileMetadata class C#

Medium

`Terrasoft.File.Abstractions.Metadata` namespace.

The `Terrasoft.File.Abstractions.Metadata.FileMetadata` abstract class provides the properties of file metadata and methods for metadata management,

Properties

`Name` `string`

File name.

`Length` `long`

The file size in bytes.

`CreatedOn` `DateTime`

Date and time the file was created.

`ModifiedOn` `DateTime`

Date and time the file was modified.

`FileContentStorageId` `Guid`

The identifier of the file storage.

`StoringState` `FileStoringState`

The state of the file (“New,” “Modified,” “Unmodified,” “Deleted”).

Methods

`abstract void SetAttribute<TValue>(string name, TValue value)`

Set the additional `name` file attribute to the specified `value`.

`abstract TValue GetAttribute<TValue>(string name, TValue defaultValue)`

Returns the specified value or the `defaultValue` of the additional `name` attribute .

`void SetStoringState(FileStoringState newState)`

Sets the file state to `FileStoringState.Modified` if the previous file state is not `FileStoringState.New`.

FileUtils class C#

Medium

`Terrasoft.File.Abstractions` namespace.

The `Terrasoft.File.Abstractions.FileUtils` class provides extension methods for file management.

Methods

`static void SetAttributes(this IFile source, IReadOnlyDictionary<string, object> attributes)`

Sets the file attributes to the values passed in the `attributes` collection.

`static void Save(this IFile source)`

Saves the metadata of the file.

`static Stream Read(this IFile source)`

Reads the content of the file.

`static void Write(this IFile source, Stream stream, FileWriteOptions writeOptions) static void W`

Writes the content of the file.

Parameters

`source`	The file whose contents is to be written
`stream`	The stream that provides the file content.
`writeOptions`	Parameters for writing the file.
`content`	The file content as an array of bytes.

`static void Delete(this IFile source)`

Deletes the specified file.

`static void Copy(this IFile source, IFile target)`

Copies the existing `source` file to the new `target` file.

`static void Move(this IFile source, IFile target)`

Moves the existing `source` file to the new `target` destination.