Namespace EdaIntegrationContract

Classes

EdaApiException

Exception generated from the Integration Library.

MigrationsRequiredException

Exception generated from the Integration Library when Case migrations are required before opening the case.

ProgramUpdateRequiredException

Exception generated from the Integration Library when program update is required before opening the case.

Interfaces

ICaseManager

Manager of all interactions for case-related data

IEdaIntegration

The initial starting point for programmatic access to the LAW and Explore functionality.

IEdaIntegrationEntity

Common interface for entities

IEdaIntegrationUniqueEntity

Common interface for unique entities

IExploreIntegration

Programmatic access to the EDA functionality used for processing data into Explore

ITurboImportIntegration

Programmatic access to the TurboImport functionality used for processing data into LAW.

Class EdaApiException

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Exception generated from the Integration Library.

```
public class EdaApiException : Exception, ISerializable
```

Inheritance

object ← Exception ← EdaApiException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

The innerException property contains additional details about the underlying error or issue that caused the failure.

Constructors

EdaApiException(string)

An exception that occurred within the Integration Library

```
public EdaApiException(string message)
```

Parameters

`message` string

A description of the error

EdaApiException(string, Exception)

An exception that occurred within the Integration Library

```
public EdaApiException(string message, Exception innerException)
```

Parameters

`message` string

A description of the error

`innerException` Exception

The original exception that caused the error

Interface ICaseManager

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Manager of all interactions for case-related data

```
public interface ICaseManager
```

Methods

All()

Retrieves a list of all the active cases

```
IEnumerable<ICaseListing> All()
```

Returns

IEnumerable<ICaseListing>

A list of ICase

Examples

The following example demonstrates retrieving the list of cases and printing each case name to the system console.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        Console.WriteLine("Name                                    Client
Description");
```

```
        Console.WriteLine("------------------------------  -------------------------  ------
-------------------");

        foreach (ICaseListing caseListing in edaIntegration.Cases.All())
        {
            Console.WriteLine("{0}  {1}  {2}",
                caseListing.Name.Substring(0, caseListing.Name.Length > 30 ? 30 :
caseListing.Name.Length).PadRight(30),
                caseListing.Client.Substring(0, caseListing.Client.Length > 30 ? 30 :
caseListing.Client.Length).PadRight(30),
                caseListing.Description.Substring(0, caseListing.Description.Length > 30 ?
30 : caseListing.Description.Length).PadRight(30)
            );
        }
    }
}
/*
This example produces the following results:
   Name                            Client                     Description
   ------------------------------  -------------------------  -------------------------
   Case 1                          Client 123                 The first case
   Case 2                          Client 456                 The second case
   Case 3                          Client 123                 The third case
 */
```

CaseExistsByName(string)

Checks if a case by this name already exists

```
bool CaseExistsByName(string caseName)
```

Parameters

caseName string

The name of the case to be checked

Returns

bool

True if a case with this name already exists

Create(string, string, string, string, bool, string, Guid?)

Creates a new case

```
ICase Create(string caseName, string caseDirectory = "", string client = "", string
description = "", bool applyDefaultFileTypeFilters = false, string databaseConnectionString
= "", Guid? caseId = null)
```

Parameters

`caseName` string

The name of the case to create

`caseDirectory` string

(optional) The location to store case-related information. By default the system will place the case directory under the $HOME/cases folder

`client` string

(optional) The name of the client to associate with this case

`description` string

(optional) A description to use for the case

`applyDefaultFileTypeFilters` bool

(optional) Flag indicating whether default file type filters will be applied for the newly created case.

`databaseConnectionString` string

(optional) A database connection string to the case database

`caseId` Guid?

(optional) The unique identifier to use to identify this case. By default a new value will be auto-generated.

Returns

ICase

A list of *ICases*

Examples

This example demonstrates how to create a new case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Values to use for the new case
        string myCaseName = @"My New Case";

        // Create the new case
        ICase edaCase = edaIntegration.Cases.Create(myCaseName);

        //Display the returned case
        Console.WriteLine("Id: " + edaCase.Name);
        Console.WriteLine("Name: " + edaCase.Name);
        Console.WriteLine("Directory: " + edaCase.CaseDirectory);
    }
}
/*
This example produces the following results:

   Id: f5c3d706-2d5b-4966-b9af-55bb43f3190f
   Name: My New Case
   Directory: \\NetworkDrive\EdAnalyzer\Home\cases\My New Case
 */
```

Delete(Guid, bool)

Deletes a case. This will delete the database and optionally the associated home directory for this case.

The amount of time it will take for this operation to complete will vary based on the whether the database is in use and the number and size of files in the Home Directory if the directory is being removed as well.

IMPORTANT

This feature is intended for use in testing with the EdaIntegration Library. Please exercise caution in its use because there is no undo function for this action.

```
void Delete(Guid caseId, bool removeHomeDirectory = true)
```

Parameters

`caseId` Guid

The unique identifier of the case to be deleted

`removeHomeDirectory` bool

Flag to indicate if the home directory should also be removed

Examples

This example demonstrates deleting a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 2");
        edaIntegration.Cases.Delete(edaCase.Id);
    }
}
```

OpenCaseById(Guid, bool)

Retrieves a specific case by its unique identifier

```
ICase OpenCaseById(Guid caseId, bool runMigrations = false)
```

Parameters

`caseId` Guid

The unique identifier of the case

`runMigrations` bool

Indicates whether to run any schema/data migrations that are needed when opening the case.

Returns

ICase

A case object

Examples

This example demonstrates retrieving a specific case by its id and printing its details to the system console.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        string caseId = "0504c9ad-bb89-445a-a2f4-045cfb459db2";
        ICase edaCase = edaIntegration.Cases.OpenCaseById(caseId);

        Console.WriteLine("Name: " + edaCase.Name);
        Console.WriteLine("Client: " + edaCase.Client);
        Console.WriteLine("Description: " + edaCase.Description);
        Console.WriteLine("Last Accessed: " + edaCase.LastAccessed);
    }
}
/*
This example produces the following results:

   Name: Case 1
   Client: Client 123
   Description:
   Last Accessed: 6/27/2014 10:59:30 PM
 */
```

Exceptions

ProgramUpdateRequiredException

MigrationsRequiredException

OpenCaseByName(string, bool)

Retrieves a specific case by its name

```
ICase OpenCaseByName(string caseName, bool runMigrations = false)
```

Parameters

`caseName` string

The name of the case

`runMigrations` bool

Indicates whether to run any schema/data migrations that are needed when opening the case.

Returns

ICase

A case object

Examples

This example demonstrates retrieving a specific case by its name

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        string caseName = "Case 1";
        ICase edaCase = edaIntegration.Cases.OpenCaseByName(caseName);
        Console.WriteLine("Name: " + edaCase.Name);
        Console.WriteLine("Client: " + edaCase.Client);
        Console.WriteLine("Description: " + edaCase.Description);
        Console.WriteLine("Last Accessed: " + edaCase.LastAccessed);
    }
}
/*
This example produces the following results:

   Name: Case 1
   Client: Client 123
   Description:
   Last Accessed: 6/27/2014 10:59:30 PM
 */
```

Exceptions

ProgramUpdateRequiredException

MigrationsRequiredException

Update(ICase)

Updates the case with any changed values

```
void Update(ICase theCase)
```

Parameters

`theCase` ICase

The case containing the updated data to save

Examples

This example demonstrates how to modify descriptive information about a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Open the case that needs to be modified.
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 2");

        // Update the information on the case that needs to be changed and save it.
        edaCase.Description = "An updated description for case 2";
        edaCase.Client = "Client 54";
        edaIntegration.Cases.Update(edaCase);

        // Re-retrieve the data from the database
        ICase edaCase2 = edaIntegration.Cases.OpenCaseByName("Case 2");

        // Display the updated values.
        Console.WriteLine("Name: " + edaCase2.Name);
        Console.WriteLine("Client: " + edaCase2.Client);
        Console.WriteLine("Description: " + edaCase2.Description);
        Console.WriteLine("Last Accessed: " + edaCase2.LastAccessed);
    }
}
/*
This example produces the following results:

   Name: Case 2
   Client: Client 54
   Description: An updated description for case 2
   Last Accessed: 6/29/2014 09:23:12 AM
 */
```

Interface IEdaIntegration

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

The initial starting point for programmatic access to the LAW and Explore functionality.

```
public interface IEdaIntegration
```

Methods

ConnectToExplore(string)

Initialize the integration library for use within the Explore environment

```
IExploreIntegration ConnectToExplore(string mgmtDbConnectionString = null)
```

Parameters

`mgmtDbConnectionString` string

The connection information to the Management database configured for Explore

Returns

IExploreIntegration

The root starting point for integrating with Explore cases

ConnectToLawTurboImport(string)

Initialize the integration library for use with LAW Turbo Import cases

```
ITurboImportIntegration ConnectToLawTurboImport(string mgmtDbConnectionString = null)
```

Parameters

`mgmtDbConnectionString` string

The connection information to the Management database configured for Explore

Returns

ITurboImportIntegration

The root starting point for integrating with LAW Cases using Turbo Import

Interface IEdaIntegrationEntity

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Common interface for entities

```
public interface IEdaIntegrationEntity
```

Interface IEdaIntegrationUniqueEntity

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Common interface for unique entities

```
public interface IEdaIntegrationUniqueEntity : IEdaIntegrationEntity
```

Properties

Id

Id field common to all entities

```
int Id { get; }
```

Property Value

int

Interface IExploreIntegration

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Programmatic access to the EDA functionality used for processing data into Explore

```
public interface IExploreIntegration
```

Properties

Cases

Access case-related data

```
ICaseManager Cases { get; }
```

Property Value

ICaseManager

Manager for case-related interactions

Interface ITurboImportIntegration

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Programmatic access to the TurboImport functionality used for processing data into LAW.

```
public interface ITurboImportIntegration : IExploreIntegration
```

Inherited Members

IExploreIntegration.Cases

Class MigrationsRequiredException

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Exception generated from the Integration Library when Case migrations are required before opening the case.

```
public class MigrationsRequiredException : Exception, ISerializable
```

Inheritance

object ← Exception ← MigrationsRequiredException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

The innerException property contains additional details about the underlying error or issue that caused the failure.

Constructors

MigrationsRequiredException(string)

Exception generated from the Integration Library when Case migrations are required before opening the case.

```
public MigrationsRequiredException(string message)
```

Parameters

`message` string

A description of the error

MigrationsRequiredException(string, Exception)

Exception generated from the Integration Library when Case migrations are required before opening the case.

```
public MigrationsRequiredException(string message, Exception innerException)
```

Parameters

`message` string

A description of the error

`innerException` Exception

The original exception that caused the error

Class ProgramUpdateRequiredException

Namespace: EdaIntegrationContract

Assembly: EdaIntegration.Contract.dll

Exception generated from the Integration Library when program update is required before opening the case.

```
public class ProgramUpdateRequiredException : Exception, ISerializable
```

Inheritance

object ← Exception ← ProgramUpdateRequiredException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

The innerException property contains additional details about the underlying error or issue that caused the failure.

Constructors

ProgramUpdateRequiredException(string)

Exception generated from the Integration Library when program update is required before opening the case.

```
public ProgramUpdateRequiredException(string message)
```

Parameters

`message` string

A description of the error

ProgramUpdateRequiredException(string, Exception)

Exception generated from the Integration Library when program update is required before opening the case.

```
public ProgramUpdateRequiredException(string message, Exception innerException)
```

Parameters

`message` string

A description of the error

`innerException` Exception

The original exception that caused the error

Namespace EdaIntegrationContract.Case

Interfaces

ICase

The basic unit of organization for identifying sets of source documents, applying filters, exporting, and other functions related to pre-filtering.

ICaseListing

The basic unit of organization for identifying sets of source documents

IGeneralCaseProperties

Access to selected case properties that don't belong to any other properties class (e.g. EmailThreadingProperties)

Enums

CaseProcessingType

The type of processing performed by Explore agents

Enum CaseProcessingType

Namespace: EdaIntegrationContract.Case

Assembly: EdaIntegration.Contract.dll

The type of processing performed by Explore agents

```
public enum CaseProcessingType
```

Fields

`CloudNineReview = 2`

Process the data into CloudNine Review

`Eda = 0`

Process the data into Explore

`TurboImport = 1`

Process the data into LAW using Turbo Import

Interface ICase

Namespace: EdaIntegrationContract.Case

Assembly: EdaIntegration.Contract.dll

The basic unit of organization for identifying sets of source documents, applying filters, exporting, and other functions related to pre-filtering.

```
public interface ICase
```

Properties

CaseDirectory

Directory where all case data is stored including search index, content storage and diagnostic logs.

```
string CaseDirectory { get; }
```

Property Value

string

Client

The client name to be associated with the case.

```
string Client { get; set; }
```

Property Value

string

Remarks

The client can be a maximum of 50 characters. Any name longer than that will be truncated to 50 characters.

Created

The date and time that the case was created

```
DateTime? Created { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Custodians

Provides access to the Custodians of this case.

```
ICustodianManager Custodians { get; }
```

Property Value

ICustodianManager

Returns the *CustodianManager* for accessing information relating to the custodians of this case.

DateRangeFilter

Provides accress to the date range filters of this case

```
IDateRangeFilterManager DateRangeFilter { get; }
```

Property Value

IDateRangeFilterManager

Returns the *DateRangeFilterManager* for accessing information relating to the date range filters of this case.

Deduplication

Provides access to Deduplication for this case.

```
IDeduplicationManager Deduplication { get; }
```

Property Value

IDeduplicationManager

Returns the *DeduplicationManager* for accessing information relating to deduplication for this case.

Description

An optional description associated with the case

```
string Description { get; set; }
```

Property Value

string

Documents

Provides access to the Documents imported into the case.

```
IDocumentManager Documents { get; }
```

Property Value

IDocumentManager

Returns the *DocumentManager* for this case.

EmailThreading

Provides access to the Email Threading manager actions for this case.

```
IEmailThreadingManager EmailThreading { get; }
```

Property Value

IEmailThreadingManager

Returns the *EmailThreadingManager* for accessing information relating to the email threading processing for this case.

Exceptions

Provides access to the ExceptionItems generated during document processing for this case.

```
IExceptionManager Exceptions { get; }
```

Property Value

IExceptionManager

Returns the *ExceptionManager* for this case.

Exports

Provides access to the Exports for this case.

```
IExportManager Exports { get; }
```

Property Value

IExportManager

Returns the *ExportManager* for accessing information relating to the export processing for this case.

FileTypeFilter

Provides access to file type filtering for this case.

```
IFileTypeFilterManager FileTypeFilter { get; }
```

Property Value

IFileTypeFilterManager

Id

A unique identifier for the case

```
Guid Id { get; }
```

Property Value

Guid

Imports

Provides access to the Import Sessions for this case.

```
IImportManager Imports { get; }
```

Property Value

IImportManager

Returns the *ImportManager* for accessing information relating to the import processing for this case.

Index

Provides access to Indexing for this case.

```
IIndexManager Index { get; }
```

Property Value

IIndexManager

Returns the *IndexManager* for accessing information relating to the indexing for this case.

IsActive

Set or get the active or inactive state for a case, used to help distinguish between active cases and inactive cases.

```
bool IsActive { get; set; }
```

Property Value

bool

Remarks

New cases are created as active = true by default. An inactive case will not be accessible via the Integration Library.

LastAccessed

The date and time that the case was last accessed in CloudNine Explore.

```
DateTime? LastAccessed { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Name

The name of the case

```
string Name { get; }
```

Property Value

string

Remarks

The name can be a maximum of 50 characters. Any name longer than that will be truncated to 50 characters.

NearDuplicates

Provides access to the Near Duplicate manager actions for this case.

```
INearDuplicateManager NearDuplicates { get; }
```

Property Value

INearDuplicateManager

Returns the *NearDuplicateManager* for accessing information relating to the near-duplicate identification processing for this case.

Nist

Provides access to Nist for this case.

```
INistManager Nist { get; }
```

Property Value

INistManager

Returns the *NistManager* for accessing information relating to Nist for this case.

OCR

Provides access to the OCR actions for this case.

```
IOcrManager OCR { get; }
```

Property Value

IOcrManager

Returns the *OcrManager* for accessing information relating to the OCR processing for this case.

ProcessingStatus

Status associated with the case

```
ProcessingStatus ProcessingStatus { get; }
```

Property Value

ProcessingStatus

Properties

Provides access to selected case properties

```
IGeneralCaseProperties Properties { get; }
```

Property Value

IGeneralCaseProperties

A proxy for selected case properties

Sources

Provides access to the Sources for this case.

```
ISourceManager Sources { get; }
```

Property Value

ISourceManager

Returns the *SourceManager* for accessing information relating to the document sources of this case.

Statistics

Gets case level aggregate statistics.

```
IStatisticsManager Statistics { get; }
```

Property Value

IStatisticsManager

Tags

Provides access to the Tags for this case.

```
ITagManager Tags { get; }
```

Property Value

ITagManager

Returns the *TagManager* for accessing information relating to the tags for this case.

Methods

GetPendingActivities()

Gets pending activities for case

```
IEnumerable<string> GetPendingActivities()
```

Returns

IEnumerable<string>

List of strings representing pending activities

UpdateProperties()

Saves the case properties in the proxy class to the EdaCase

```
void UpdateProperties()
```

Interface ICaseListing

Namespace: EdaIntegrationContract.Case

Assembly: EdaIntegration.Contract.dll

The basic unit of organization for identifying sets of source documents

```
public interface ICaseListing
```

Properties

CaseDirectory

Directory where all case data is stored including search index, content storage and diagnostic logs.

```
string CaseDirectory { get; }
```

Property Value

string

Client

The client name associated with the case.

```
string Client { get; }
```

Property Value

string

Created

The date and time that the case was created

```
DateTime? Created { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Description

An optional description associated with the case

```
string Description { get; }
```

Property Value

string

Id

A unique identifier for the case

```
string Id { get; }
```

Property Value

string

LastAccessed

The date and time that the case was last accessed in CloudNine Explore.

```
DateTime? LastAccessed { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

MigrationsRequired

Indicates whether database schema and possibly data migrations are required when opening this case.

```
bool MigrationsRequired { get; }
```

Property Value

bool

Name

The name of the case

```
string Name { get; }
```

Property Value

string

ProgramUpdateRequired

Indicates whether program update is require when opening this case.

```
bool ProgramUpdateRequired { get; }
```

Property Value

bool

Interface IGeneralCaseProperties

Namespace: EdaIntegrationContract.Case

Assembly: EdaIntegration.Contract.dll

Access to selected case properties that don't belong to any other properties class (e.g. EmailThreadingProperties)

```
public interface IGeneralCaseProperties
```

Properties

LawCaseDir

Location of the LAW case (used for Turbo Import)

```
string LawCaseDir { get; set; }
```

Property Value

string

LockCaseSettings

Do not allow users to change certain settings once data is imported into the case.

```
bool LockCaseSettings { get; }
```

Property Value

bool

TimeZoneId

The case's timezone. Default value of empty string indicates no selection.

```
string TimeZoneId { get; set; }
```

Property Value

string

Namespace EdaIntegrationContract.Document

Interfaces

IAttachmentMetadata

Document attachments metadata

IDocument

Interface to metadata for a document

IDocumentManager

Responsible for retrieving documents from a case

IDupeMetadata

Document metadata regarding Duplicate analysis

IEdocMetadata

Metadata specific to electronic documents

IEmailMetadata

Metadata specific to Email and calendar items

Enums

DocDateSource

EDocument date created and date modified source

DocumentType

Identifies the type of document

Enum DocDateSource

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

EDocument date created and date modified source

```
public enum DocDateSource : byte
```

Fields

`ArchiveEntry = 4`

Date created and date modified were obtained from the document archive entry

`FileSystem = 1`

Date created and date modified were obtained from the file system dates

`MapiProperties = 3`

Date created and date modified were obtained from the parent message properties

`Metadata = 5`

Date created and date modified were obtained from the document metadata

`None = 0`

Date created and date modified were not obtained

`Parent = 2`

Date created and date modified were inherited from the parent document

Enum DocumentType

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

Identifies the type of document

```
public enum DocumentType
```

Fields

`EDocument = 0`

Electronic document

`Email = 1`

Email document

Interface IAttachmentMetadata

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

Document attachments metadata

```
public interface IAttachmentMetadata
```

Properties

AttachmentExportNumbers

Populated for parent records only, this field contains a list of export numbers for the parent and each attachment record. Ex: A0001 for the parent, A0002 and A0003 for its attachments.

```
ICollection<string> AttachmentExportNumbers { get; }
```

Property Value

ICollection<string>

AttachmentNames

list of file names of the (top-level) attached files

```
ICollection<string> AttachmentNames { get; }
```

Property Value

ICollection<string>

AttachmentParentExportNumber

Export number of the (top-level) parent record. Applied to all members of the family. Ex: A0001

```
string AttachmentParentExportNumber { get; }
```

Property Value

string

AttachmentRangeExportNumbers

Export numbers of the parent record and the last attachment in the family, separated by a hyphen. Field is empty for records that are not part of a parent/attachment relationship. Ex: A0001 - A0003

```
string AttachmentRangeExportNumbers { get; }
```

Property Value

string

Interface IDocument

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

Interface to metadata for a document

```
public interface IDocument
```

Properties

CompositeName

A concatenated value used to describe the location of a document within its container.

```
string CompositeName { get; }
```

Property Value

string

CompressedSize

The compressed size of the native file (in bytes) if it was contained within certain archive types (e.g. not populated for 7zip items).

```
long CompressedSize { get; }
```

Property Value

long

ContainerPath

If the file was originally contained within a container, e.g. .zip file or .pst, this is the path to the file's location within the container.

```
string ContainerPath { get; }
```

Property Value

string

Custodian

The custodian responsible for the document.

```
ICustodian Custodian { get; }
```

Property Value

ICustodian

DateFilterLowerBound

For emails, this field contains the sent date of the message. For calendar / appointment items, this holds the appointment start date and time. For e-documents, the earlier of the two dates found in the creation and last modified fields (within the internal document metadata (if available)) is used.

```
DateTime? DateFilterLowerBound { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

DateFilterUpperBound

For emails, this field contains the sent date of the message. For calendar / appointment items, this holds the appointment end date and time. For e-documents, the latest of the two dates found in the creation and last modified fields (within the internal document metadata (if available)) is used.

```
DateTime? DateFilterUpperBound { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Decrypted

Indicates if the native file was password protected but able to be decrypted during import.

```
bool Decrypted { get; }
```

Property Value

bool

DedupeStatus

Indicates if the document was identified as a duplicate of another document in the case during Analysis. See DeduplicationStatus for possible values.

```
DeduplicationStatus DedupeStatus { get; }
```

Property Value

DeduplicationStatus

Directory

The path to the original folder/location of the native file. For each item that originated in a mail store, the Directory will list the path to the mail store.

```
string Directory { get; }
```

Property Value

string

DocExtension

The original document extension unless "Assign Suspect Extensions" case setting is on and File Type Management (FTM) has a better extension for the identified file type.

```
string DocExtension { get; }
```

Property Value

string

DocumentType

The type of document that this represents, e.g. Email or EDocument.

```
DocumentType DocumentType { get; }
```

Property Value

DocumentType

EDocMetadata

Metadata specific to non-email documents. For Email documents this will be Null.

```
IEdocMetadata EDocMetadata { get; }
```

Property Value

IEdocMetadata

EmailMetadata

Metadata specific to Email documents. For non-email documents this will be Null.

```
IEmailMetadata EmailMetadata { get; }
```

Property Value

IEmailMetadata

Encrypted

Indicates if the native file was password protected and unable to be decrypted.

```
bool Encrypted { get; }
```

Property Value

bool

FamilyParentId

The unique identifier of the "family parent" document, i.e. the "top" document in the document family

```
Guid FamilyParentId { get; }
```

Property Value

Guid

FileName

The original filename and extension of the imported native file.

```
string FileName { get; }
```

Property Value

string

FileType

A numeric representation of the native file type (provided by third-party file identification engine).

```
int FileType { get; }
```

Property Value

int

FileTypeDescription

A textual description of native file type (provided by third-party file identification engine).

```
string FileTypeDescription { get; }
```

Property Value

string

FileUri

Used internally by the system to retrieve mail store items for exporting and viewing native files, e.g. (Lotus Notes email): note-id://D2D09967C7679B3E8625711400748A11.

```
string FileUri { get; }
```

Property Value

string

HasAttachments

Indicates whether the document has attachments.

```
bool HasAttachments { get; }
```

Property Value

bool

Hash

The MD5 or SHA-1 hash of the email or edocument, depending on the Deduplication Mode setting at the time of import.

```
string Hash { get; }
```

Property Value

string

Id

The unique identifier for the document.

```
Guid Id { get; }
```

Property Value

Guid

ImportSet

The document's import session.

```
IImportSet ImportSet { get; }
```

Property Value

IImportSet

Imported

The date/time that the file was imported into the case.

```
DateTime? Imported { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC.

InventoryId

The internal numeric identifier for the document.

```
int InventoryId { get; }
```

Property Value

int

IsArchiveItem

Indicates whether the native file was contained with an archive (e.g. 7zip) when imported into the case.

```
bool IsArchiveItem { get; }
```

Property Value

bool

IsAttachment

Indicates whether the native file was an attachment of another document.

```
bool IsAttachment { get; }
```

Property Value

bool

IsCompound

Indicates whether the native file is a "compound document" (contains embedded files).

```
bool IsCompound { get; }
```

Property Value

bool

IsCompoundChild

Indicates whether the native file was embedded inside another document (the parent document would have IsCompound set to True).

```
bool IsCompoundChild { get; }
```

Property Value

bool

IsEmailAttachment

Indicates whether the native file was an attachment of an email.

```
bool IsEmailAttachment { get; }
```

Property Value

bool

IsSuspectExtension

True if "Assign Suspect Extensions" case setting is on and File Type Management (FTM) has a better extension for the identified file type, false otherwise.

```
bool IsSuspectExtension { get; }
```

Property Value

bool

IsTextStoredOnDisk

Indicates if the text of the document is being returned in the Text or if the text was stored on disk. If it is on disk then the TextLocation will contain the path the file.

```
bool IsTextStoredOnDisk { get; }
```

Property Value

bool

Level

A numeric representation of the depth of the document within its container. For example, a PST file might be level 0, an email in the PST level 1, an attachment to an email in the PST level 2, etc.

```
int Level { get; }
```

Property Value

int

MimeType

Information about the content of the file extracted from the header of emails (with the exception of MS Outlook items) and e-documents. Not all e-documents contain this information.

```
string MimeType { get; }
```

Property Value

string

ParentId

The unique identifier (Id) of the immediate parent document in the parent-child relationship hierarchy, e.g. parent doc or mail store.

```
Guid ParentId { get; }
```

Property Value

Guid

Password

The password used to decrypt the native file during import.

```
string Password { get; }
```

Property Value

string

Size

The size of the imported/analyzed native file in bytes.

```
long Size { get; }
```

Property Value

long

Source

The source of the document.

```
ISource Source { get; }
```

Property Value

ISource

SourceDocumentId

The unique identifier (Id) of the top-level document in the parent-child relationship hierarchy, e.g. parent doc or mail store.

```
Guid SourceDocumentId { get; }
```

Property Value

Guid

SourceFile

The full path to the native file. If the file was in a container then the full path to the top-most container is returned.

```
string SourceFile { get; }
```

Property Value

string

Text

The text of the document extracted during Analysis or OCR. If the text was stored on disk (IsTextStoredOnDisk) then this value will be null.

```
string Text { get; }
```

Property Value

string

TextLocation

The full path to the location of the document text if it was stored on disk. If the text was not stored on disk (IsTextStoredOnDisk) then this value will be null.

```
string TextLocation { get; }
```

Property Value

string

TextSize

Size of the extracted or OCR text (in bytes).

```
long TextSize { get; }
```

Property Value

long

Interface IDocumentManager

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

Responsible for retrieving documents from a case

```
public interface IDocumentManager
```

Methods

ById(IEnumerable<Guid>)

Retrieves a set of documents matching a list of unique identifiers

```
IEnumerable<IDocument> ById(IEnumerable<Guid> documentIds)
```

Parameters

`documentIds` IEnumerable<Guid>

The unique identifiers of the documents to find

Returns

IEnumerable<IDocument>

A list of documents matching the supplied unique identifiers

Examples

The following example demonstrates retrieving multiple documents using a list of ids.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Document;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Retrieve all documents that have exceptions
        int[] docIds = edaCase.Exceptions.All().Select(x => x.DocumentId).ToArray();
        IDocument[] docs = edaCase.Documents.ById(docIds).ToArray();

        // Print the document id and name
        Console.WriteLine("All documents with exceptions");
        Console.WriteLine("=============================");
        foreach (IDocument doc in docs)
        {
            if (doc.DocumentType == DocumentType.EDocument)
            {
                Console.WriteLine("{0,6}  {1, -25}", doc.Id, doc.FileName);
            }
            else if (doc.DocumentType == DocumentType.Email)
            {
                Console.WriteLine("{0,6}  {1, -25}", doc.Id, doc.EmailMetadata.Subject);
            }
        }
    }
}
/*
This example produces the following results:

All documents with exceptions
======================================
    26  Doc1_metadata.docm
     6  Empty Content.rtf
    22  early.docx
    15  lorem_pw.docx
     3  Custom Encrypted.pptx
     1  Background.jpg
     7  nashvile march.jpg
*/
```

ById(Guid)

Retrieves a Document by its unique identifier

```
IDocument ById(Guid id)
```

Parameters

id Guid

The unique identifier of the document to find

Returns

IDocument

The document with the supplied unique identifier

Examples

The following example demonstrates retrieving a document using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Document;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IDocument doc = edaCase.Documents.ById(15);
        Console.WriteLine("Id: {0}", doc.Id);
        if (doc.DocumentType == DocumentType.EDocument)
        {
            Console.WriteLine("File Name: {0}", doc.FileName);
        }
        else if (doc.DocumentType == DocumentType.Email)
        {
            Console.WriteLine("Subject: {0}", doc.EmailMetadata.Subject);
        }
    }
}
/*
This example produces the following results:
```

```
Id: 15
Name: lorem_pw.docx
 */
```

Exceptions

EdaApiException

Thrown if a document cannot be found with the specified Id

ByInventoryId(IEnumerable<int>)

Retrieves a set of documents matching a list of unique identifiers

```
IEnumerable<IDocument> ByInventoryId(IEnumerable<int> documentIds)
```

Parameters

`documentIds` IEnumerable<int>

The unique identifiers of the documents to find

Returns

IEnumerable<IDocument>

A list of documents matching the supplied unique identifiers

Examples

The following example demonstrates retrieving multiple documents using a list of ids.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Document;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Retrieve all documents that have exceptions
        int[] docIds = edaCase.Exceptions.All().Select(x => x.DocumentId).ToArray();
        IDocument[] docs = edaCase.Documents.ById(docIds).ToArray();

        // Print the document id and name
        Console.WriteLine("All documents with exceptions");
        Console.WriteLine("=============================");
        foreach (IDocument doc in docs)
        {
            if (doc.DocumentType == DocumentType.EDocument)
            {
                Console.WriteLine("{0,6}  {1, -25}", doc.Id, doc.FileName);
            }
            else if (doc.DocumentType == DocumentType.Email)
            {
                Console.WriteLine("{0,6}  {1, -25}", doc.Id, doc.EmailMetadata.Subject);
            }
        }
    }
}
/*
This example produces the following results:

All documents with exceptions
======================================
    26  Doc1_metadata.docm
     6  Empty Content.rtf
    22  early.docx
    15  lorem_pw.docx
     3  Custom Encrypted.pptx
     1  Background.jpg
     7  nashvile march.jpg
*/
```

ByInventoryId(int)

Retrieves a Document by its unique identifier

```
IDocument ByInventoryId(int inventoryId)
```

Parameters

inventoryId int

The unique identifier of the document to find

Returns

IDocument

The document with the supplied unique identifier

Examples

The following example demonstrates retrieving a document using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Document;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IDocument doc = edaCase.Documents.ById(15);
        Console.WriteLine("Id: {0}", doc.Id);
        if (doc.DocumentType == DocumentType.EDocument)
        {
            Console.WriteLine("File Name: {0}", doc.FileName);
        }
        else if (doc.DocumentType == DocumentType.Email)
        {
            Console.WriteLine("Subject: {0}", doc.EmailMetadata.Subject);
        }
    }
}
/*
This example produces the following results:

Id: 15
Name: lorem_pw.docx
 */
```

Exceptions

EdaApiException

Thrown if a document cannot be found with the specified Id

Delete(IEnumerable<Guid>)

Deletes one or more documents

WARNING

When deleting a document, the metadata and text associated with that document and all of its children will be deleted. This action is not recoverable.

```
void Delete(IEnumerable<Guid> documentIds)
```

Parameters

`documentIds` IEnumerable<Guid>

The list of unique identifiers of the documents to be deleted

Examples

The following example demonstrates deleting a single document from the case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Document;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // The identifier of the document(s) would be known by some other mechanism,
        // perhaps through a separate query, exception information, etc.
        var docsToDelete = new Guid[] { Guid.Parse("a950c8cd-9ec8-4d62-9689-
032cf5290672") };

        // Use that identifier to delete the document.
        edaCase.Documents.Delete(docsToDelete);
    }
}
```

Delete(IEnumerable<int>)

Deletes one or more documents

WARNING

When deleting a document, the metadata and text associated with that document and all of its children will be deleted. This action is not recoverable.

```
void Delete(IEnumerable<int> inventoryIds)
```

Parameters

`inventoryIds` IEnumerable<int>

The list of identifiers of the documents to be deleted

Examples

The following example demonstrates deleting a single document from the case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Document;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        // The inventoryId of the document(s) would be known by some other mechanism,
        // perhaps through a separate query, exception information, etc.
        var docsToDelete = new int[] { 668428 };

        // Use that identifier to delete the document.
        edaCase.Documents.Delete(docsToDelete);
    }
}
```

Interface IDupeMetadata

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

Document metadata regarding Duplicate analysis

```
public interface IDupeMetadata
```

Properties

DuplicateCustodianNames

Names of the custodians containing duplicate versions of the original record. Populated for parent duplicates/original records only.

```
ICollection<string> DuplicateCustodianNames { get; }
```

Property Value

ICollection<string>

DuplicateCustodianPaths

Source path to each duplicate version of the original record. Populated for parent/original records only. e-doc ex: CustName\Folder01\Folder02 email ex: CustName\Mail-Stores\Archive.pst\Inbox

```
ICollection<string> DuplicateCustodianPaths { get; }
```

Property Value

ICollection<string>

DuplicateParentExportNumber

Contains the export number of the original record in a duplicate relationship (the original and all associated duplicates are assigned the same value).

```
string DuplicateParentExportNumber { get; }
```

Property Value

string

DuplicateParentName

Custodian name of the original record. Populated for duplicate records only.

ex: ParentCustodianName

```
string DuplicateParentName { get; }
```

Property Value

string

DuplicateParentPath

Source path of the original record. Populated for duplicate records only.

e-doc ex: CustName\Folder01\Folder02

email ex: CustName\Mail-Stores\Archive.pst\Inbox

```
string DuplicateParentPath { get; }
```

Property Value

string

Interface IEdocMetadata

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

Metadata specific to electronic documents

```
public interface IEdocMetadata
```

Properties

Application

The application used to create the document.

```
string Application { get; }
```

Property Value

string

Author

The author field value extracted from the metadata of the native file during analysis, e.g. John Smith.

```
string Author { get; }
```

Property Value

string

Categories

The category field value extracted from metadata of the native file.

```
string Categories { get; }
```

Property Value

string

Comments

The comments field value extracted from the metadata of the native file.

```
string Comments { get; }
```

Property Value

string

Created

The file creation date and time extracted from the metadata of the native file during analysis.

```
DateTime? Created { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC.

DateSource

EDocument date created and date modified source. See DocDateSource for possible values.

```
DocDateSource DateSource { get; }
```

Property Value

DocDateSource

HasComments

Indicates whether comments exist in the Microsoft Word, Microsoft PowerPoint, or Microsoft Excel native file, or whether sticky notes exist in the Adobe Acrobat PDF native file.

```
bool HasComments { get; }
```

Property Value

bool

HasHiddenRowsCols

Indicates whether the native Microsoft Excel file contains hidden row(s) or column(s).

```
bool HasHiddenRowsCols { get; }
```

Property Value

bool

HasHiddenSheets

Indicates whether the native Microsoft Excel file contains hidden sheet(s).

```
bool HasHiddenSheets { get; }
```

Property Value

bool

HasHiddenSlides

Indicates whether the native Microsoft PowerPoint file contains hidden slide(s).

```
bool HasHiddenSlides { get; }
```

Property Value

bool

HasHiddenText

Indicates whether the document contains hidden text

```
bool HasHiddenText { get; }
```

Property Value

bool

HasImages

Indicates whether the native PDF document contains any images. If the document contains any type of images, it is eligible for OCR.

```
bool HasImages { get; set; }
```

Property Value

bool

HasSpeakerNotes

Indicates whether the native Microsoft PowerPoint file contains speaker note(s).

```
bool HasSpeakerNotes { get; }
```

Property Value

bool

HasTrackedChanges

Indicates whether the Microsoft Word or Microsoft Excel file contains tracked change(s).

```
bool HasTrackedChanges { get; }
```

Property Value

bool

Keywords

The keywords extracted from the metadata of the native file.

```
string Keywords { get; }
```

Property Value

string

LastAuthor

The user who last saved or revised the document, e.g. John Smith.

```
string LastAuthor { get; }
```

Property Value

string

LastModified

The last modified date and time extracted from the metadata of the native file during analysis.

```
DateTime? LastModified { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC.

LastPrinted

The date and time the document was last printed.

```
DateTime? LastPrinted { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC.

Organization

Organization or Company value retrieved from the metadata properties of the document.

```
string Organization { get; }
```

Property Value

string

Revision

The revision number of the document found in the file system properties.

```
int Revision { get; }
```

Property Value

int

Subject

The subject field value extracted from metadata of native file.

```
string Subject { get; }
```

Property Value

string

Template

The template field value extracted from the metadata of the native file during analysis, e.g. Normal.dotm.

```
string Template { get; }
```

Property Value

string

Title

The title field value extracted from the metadata of the native file.

```
string Title { get; }
```

Property Value

string

Interface IEmailMetadata

Namespace: EdaIntegrationContract.Document

Assembly: EdaIntegration.Contract.dll

Metadata specific to Email and calendar items

```
public interface IEmailMetadata
```

Properties

AppointmentEnd

The appointment end date and time for calendar/appointment items.

```
DateTime? AppointmentEnd { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

AppointmentLocation

The location where the appointment was held.

```
string AppointmentLocation { get; }
```

Property Value

string

AppointmentOptional

The list of optional recipients for the appointment.

```
string AppointmentOptional { get; }
```

Property Value

string

AppointmentOrganizer

The organizer of the appointment.

```
string AppointmentOrganizer { get; }
```

Property Value

string

AppointmentRequired

The list of required recipients for the appointment.

```
string AppointmentRequired { get; }
```

Property Value

string

AppointmentStart

The appointment start date and time for calendar/appointment items.

```
DateTime? AppointmentStart { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

AppointmentType

The type of the appointment.

```
string AppointmentType { get; }
```

Property Value

string

Attachments

A semi-colon-delimited list of filenames of the (top-level) attached files.

ex: UserGuide.pdf;HelpFile.chm

```
string Attachments { get; }
```

Property Value

string

BCC

The email addresses of the recipient(s) of "Blind Carbon Copies" of the email message, comma or semi-colon delimited (depends upon email client).

```
string BCC { get; }
```

Property Value

string

CC

The email addresses of the recipient(s) of "Carbon Copies" of the email message, comma or semi-colon delimited (depends upon email client).

```
string CC { get; }
```

Property Value

string

CodePage

A numeric value that specifies the character set used in the email.

```
long CodePage { get; }
```

Property Value

long

ConversationId

A Microsoft-specific value used to identify an e-mail conversation.

ex: 01CF5D946F4EFD65607202

```
string ConversationId { get; }
```

Property Value

string

ConversationIndex

A Microsoft-specific value used for tracking the position of an email within a conversation. This value is used during the process of email threading.

ex: 0101CF602E916BDDE035F5FB3D47A8D8D2D6F92CC769

```
string ConversationIndex { get; }
```

Property Value

string

ConversationTopic

The normalized subject of the email.

```
string ConversationTopic { get; }
```

Property Value

string

DeliveryReceipt

Indicates whether the option to request a delivery receipt was enabled in the email message.

```
bool DeliveryReceipt { get; }
```

Property Value

bool

EmailClient

The email client used to send the email message.

This information not always available.

```
string EmailClient { get; }
```

Property Value

string

EntryId

The unique identifier of the email within its mail store.

ex: 000000003DE46CC0FC76C04187C0BB41FBEBB063E4002000

```
string EntryId { get; }
```

Property Value

string

From

The email address of the sender of the email message.

```
string From { get; }
```

Property Value

string

FromDomain

The domain portion (only) of the sender's email address.

```
string FromDomain { get; }
```

Property Value

string

Headers

The contents of the header extracted from the email message

This information not always available.

```
string Headers { get; }
```

Property Value

string

IconIndex

Used by Microsoft Exchange to determine which icon to display (e.g. MailUnread).

ex: 4294967295

```
long IconIndex { get; }
```

Property Value

long

Importance

The value of the Importance flag contained in the email message.

```
int Importance { get; }
```

Property Value

int

ImportanceValue

A string representation of the email importance.

ex: High

```
string ImportanceValue { get; }
```

Property Value

string

InReplyToId

The Internet Message ID of the e-mail replied to.

```
string InReplyToId { get; }
```

Property Value

string

InternetMessageId

The Internet Message ID assigned to an e-mail message by the outgoing mail server.

```
string InternetMessageId { get; }
```

Property Value

string

MessageClass

Indicates the item / entry type in the mail store.

```
string MessageClass { get; }
```

Property Value

string

ReadReceipt

Indicates whether the option to request a delivery receipt was enabled in the email message.

```
bool ReadReceipt { get; }
```

Property Value

bool

Received

Date and time the email message was received.

```
DateTime? Received { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Sensitivity

The value of the Sensitivity flag contained in the email message.

```
int Sensitivity { get; }
```

Property Value

int

SensitivityValue

A string representation of the email sensitivity.

ex: Confidential

```
string SensitivityValue { get; }
```

Property Value

string

Sent

The sent date and time of the email message.

```
DateTime? Sent { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Subject

The subject line from the email message.

```
string Subject { get; }
```

Property Value

string

To

The email addresses of the main recipient(s) of the email message, comma or semi-colon delimited (depends upon email client).

```
string To { get; }
```

Property Value

string

UnRead

Indicates whether the email was marked read/unread.

```
bool UnRead { get; }
```

Property Value

bool

UnSent

Indicates whether the email was sent (for emails within a mail store).

```
bool UnSent { get; }
```

Property Value

bool

Namespace EdaIntegrationContract.Email Threading

Interfaces

IEmailThreadMetadata
  Document metadata concerning email threading

IEmailThreadingManager
  Represents a class for managing the email threading process.

IEmailThreadingProperties
  Represents the email threading properties for a case.

Interface IEmailThreadMetadata

Namespace: EdaIntegrationContract.EmailThreading

Assembly: EdaIntegration.Contract.dll

Document metadata concerning email threading

```
public interface IEmailThreadMetadata
```

Properties

InclusiveReason

If IsInclusive is true, this field indicates why the email is inclusive. If the email message contains message content that is not copied in any of its subsequent replies or forwards, "Message" is displayed. If the email message has attachments that are not included in any of its subsequent replies or forwards, "Attachment" is displayed. If the email message has both new message content and new attachments not included in any of its subsequent replies or forwards, "Message, Attachment" is displayed.

If IsInclusive false, this value is blank.

```
string InclusiveReason { get; }
```

Property Value

string

IsInclusive

Flags a minimal set of emails which can be viewed in order to read the email message thread's entire conversation. In the simplest case this will simply be the last message in the thread, since it will quote all the previous messages.

```
bool IsInclusive { get; }
```

Property Value

bool

IsMessage

Indicates whether the document was recognized as an email message.

```
bool IsMessage { get; }
```

Property Value

bool

Level

Level of this email message in relation to the first email in the email thread associated with this message. First email in the email thread is Level 0.

```
int Level { get; }
```

Property Value

int

MessageId

Unique ID that is assigned to each message. If several documents have the same Message ID, it is because they were recognized as separate copies of the same email message.

```
int? MessageId { get; }
```

Property Value

int?

ParentId

The immediate parent of the current email in the thread. For the first message, this will be blank. For each subsequent one, it will be the Document Id of the email which it replies to or is a forward of. For attachments, ParentID indicates the email to which it is attached.

```
int? ParentId { get; }
```

Property Value

int?

ThreadId

A unique ID assigned to each message thread.

```
int? ThreadId { get; }
```

Property Value

int?

ThreadIndex

Composed of a base ID, plus a unique number for each message within the thread, and if there are attachments, the letter A plus the unique number for the attachment within the thread. For example, A1, A2, A3 Uses the format: [base ID].[unique number for message].A[unique attachment number]. The base ID is the same as the thread ID without the leading zeros. The thread index for the root message of the thread will just be this. ex: 4.1.2.3.4.5.6.7

```
string ThreadIndex { get; }
```

Property Value

string

ThreadSize

The total number of unique messages in the thread.

```
int ThreadSize { get; }
```

Property Value

int

ThreadSort

A sort order which follows the chain of conversation, from first message to most recent. If a conversation splits into multiple branches, ThreadSort will keep each of the branches together.

```
int ThreadSort { get; }
```

Property Value

int

Interface IEmailThreadingManager

Namespace: EdaIntegrationContract.EmailThreading

Assembly: EdaIntegration.Contract.dll

Represents a class for managing the email threading process.

```
public interface IEmailThreadingManager
```

Properties

Properties

Provides access to the email threading properties for this case.

```
IEmailThreadingProperties Properties { get; }
```

Property Value

IEmailThreadingProperties

Returns the *EmailThreadingProperties* for accessing email threading related properties for this case.

Methods

UpdateProperties()

Saves the IEmailThreadingProperties to the case.

```
void UpdateProperties()
```

Examples

The following example demonstrates updating the Email Threading properties for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.EmailThreading;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the NearDupeComparisonThreshold value
        edaCase.EmailThreading.Properties.Enabled = true;

        // Save the changes to the case
        edaCase.EmailThreading.UpdateProperties();
    }
}
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IEmailThreadingProperties

Namespace: EdaIntegrationContract.EmailThreading

Assembly: EdaIntegration.Contract.dll

Represents the email threading properties for a case.

```
public interface IEmailThreadingProperties
```

Properties

EmailThreadingEnabled

Determines whether email threading analysis is enabled for the case.

```
bool EmailThreadingEnabled { get; set; }
```

Property Value

bool

Examples

This example demonstrates how to set and retrieve the EmailThreadingEnabled property.

```
using Law.EdaIntegration;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the EmailThreadingEnabled value
        edaCase.EmailThreading.Properties.EmailThreadingEnabled = true;

        // Save the changes to the case
        edaCase.EmailThreading.UpdateProperties();
        // Get the value
        Console.WriteLine("EmailThreadingEnabled: {0}",
edaCase.EmailThreading.Properties.EmailThreadingEnabled);
    }
}
/*
This example produces the following results:

EmailThreadingEnabled: True
 */
```

Remarks

When email threading analysis is enabled for a case, the email threading processing will automatically start if the system detects email threading analysis has never run on the case or if there are changes to the case's current data since the last time the analysis ran. Examples of changes that could cause email threading analysis to be run include the import of additional documents or deletion of case documents

See Also

UpdateProperties()

Namespace EdaIntegrationContract.Exceptions

Interfaces

IExceptionFilter

The filter to be applied when limiting the exceptions returned

IExceptionFilterValue

Container for returning details about what values can be filtered on and the number of exceptions matching the value

IExceptionItem

An exception that occurred during processing

IExceptionManager

The manager of exceptions that occurred during document processing in a case

IExceptionProperties

Represents properties that affect the creation of exceptions for the case.

Enums

ExceptionFilterType

The types of filters that can be used to limit the return of exception information

ExceptionType

The types of exceptions raised when processing documents

Enum ExceptionFilterType

Namespace: EdaIntegrationContract.Exceptions

Assembly: EdaIntegration.Contract.dll

The types of filters that can be used to limit the return of exception information

```
public enum ExceptionFilterType
```

Fields

`Custodian = 0`

Filter by Custodian

`FileType = 1`

Filter by the type of file processed

`ImportSet = 2`

Filter by Import Set

`Source = 3`

Filter by Source

Enum ExceptionType

Namespace: EdaIntegrationContract.Exceptions

Assembly: EdaIntegration.Contract.dll

The types of exceptions raised when processing documents

```
public enum ExceptionType
```

Fields

`AnalysisFailed = 0`

A critical failure occurred that caused the Analysis process to abort.

`AnalyticsFailed = 24`

A critical failure occurred that caused the Analytics process to abort.

`ArchiveOpenError = 1`

An error occurred attempting to parse an archive failed due to corruption, encryption or mis-identification.

`ContainerError = 2`

An error occurred attempting to list the contents of a container or sub-container.

`ContentExtractionError = 3`

An error occurred during extracting the content from a file.

`DateRange = 18`

date range exception

`DeduplicationFailed = 4`

An error occurred during duplicate detection

`DocumentDelete = 5`

Error deleting documents

`EmailOpenError = 6`

An error occurred attempting to parse a file that appears to be an email.

`EmailThreadDetectionError = 25`

An error occurred during the email threading process.

`EmptyFile = 19`

A file was successfully processed but it was an empty (zero byte) file.

`EncryptedFile = 20`

The contents of a file were unable to be processed due to encryption.

`ExecutableFile = 21`

An executable (binary) file was encountered.

`FilterError = 7`

An error occurred during evaluating a processing filter (e.g. duplicate/nist).

`ForensicImageOpenFailureError = 8`

An error occurred attempting to open or process a forensic image file.

`HashError = 9`

An error occurred during file and email hashing operations.

`IdentificationError = 10`

An error occurred during the file type identification process.

`IndexError = 26`

An error occurred during the indexing process.

`IndexingFailed = 27`

A critical failure occurred that caused the Indexing process to abort.

`IndexingTruncated = 28`

Only a portion of the content was indexed because it was too large to completely index.

`InventoryError = 11`

An error occurred during the inventory process.

`InventoryFailed = 12`

A critical failure occurred that caused the Inventory process to abort.

`LanguageRecognitionError = 13`

An error occurred identifying the language of a document.

`MailStoreOpenError = 14`

An error occurred attempting to process an inaccessible or corrupted mailstores, e.g. a PST.

`MetadataExtractionError = 15`

An error occurred during the extraction of metadata from a file.

`NearDuplicateDetectionError = 29`

An error occurred during near duplicate detection processing.

`NearDuplicateGroupingError = 30`

An error occurred during the near duplicate grouping process.

`NearNativeImagingError = 31`

Logged for Near Native Imaging errors

`NoContent = 22`

A file was successfully processed but it contained no content.

`NoOcrContent = 32`

OCR was attempted for a document but no content was retrieved.

`OcrError = 33`

An error occurred during the OCRing of a document.

`OcrFailed = 34`

A critical failure occurred that caused the OCR process to abort.

`ReprocessingRequired = 16`

Due to upgrade issues, reprocessing of the document is required in order to resolve issues with extracting the documents attachments

`Unknown = 17`

An unknown exception type was encountered.

`UnknownFileType = 23`

The file type of a document was unable to be determined.

Interface IExceptionFilter

Namespace: EdaIntegrationContract.Exceptions

Assembly: EdaIntegration.Contract.dll

The filter to be applied when limiting the exceptions returned

```
public interface IExceptionFilter
```

Properties

CustodianIds

An optional list of custodian identifiers that should be used to limit the list of exceptions returned.

```
IEnumerable<int> CustodianIds { get; set; }
```

Property Value

IEnumerable<int>

FileTypeIds

An optional list of file type identifiers that should be used to limit the list of exceptions returned.

```
IEnumerable<int> FileTypeIds { get; set; }
```

Property Value

IEnumerable<int>

ImportSetIds

An optional list of Import Set identifiers that should be used to limit the list of exceptions returned.

```
IEnumerable<int> ImportSetIds { get; set; }
```

Property Value

IEnumerable<int>

IncludeExceptionsForExcludedDocuments

Flag controlling whether exceptions for documents excluded by EDA filters are included in the results.

```
bool IncludeExceptionsForExcludedDocuments { get; set; }
```

Property Value

bool

SourceIds

An optional list of Source Ids that should be used to limit the list of exceptions returned.

```
IEnumerable<int> SourceIds { get; set; }
```

Property Value

IEnumerable<int>

Types

An optional list of exception types that should be used to limit the list of exceptions returned.

```
IEnumerable<ExceptionType> Types { get; set; }
```

Property Value

IEnumerable<ExceptionType>

Interface IExceptionFilterValue

Namespace: EdaIntegrationContract.Exceptions

Assembly: EdaIntegration.Contract.dll

Container for returning details about what values can be filtered on and the number of exceptions matching the value

```
public interface IExceptionFilterValue
```

Properties

Description

The description of the filter value, e.g. a custodian's name or description of a file type

```
string Description { get; }
```

Property Value

string

ExceptionCount

The number of exceptions matching the filter value criteria

```
int ExceptionCount { get; }
```

Property Value

int

Id

The unique identifier of the filter value

```
int Id { get; }
```

Property Value

int

Interface IExceptionItem

Namespace: EdaIntegrationContract.Exceptions

Assembly: EdaIntegration.Contract.dll

An exception that occurred during processing

```
public interface IExceptionItem : IEdaIntegrationUniqueEntity, IEdaIntegrationEntity
```

Properties

Created

The date and time that the exception was generated.

```
DateTime Created { get; }
```

Property Value

DateTime

Detail

Any additional details for the exception, e.g. an expanded description or stack trace.

```
string Detail { get; }
```

Property Value

string

DocumentId

The unique identifier of the document that generated the exception.

```
int? DocumentId { get; }
```

Property Value

int?

FileReference

The full path reference to the files location, including container name, path within the container, and parent document if these are applicable.

```
string FileReference { get; }
```

Property Value

string

Id

The unique identifier of the exception.

```
int Id { get; }
```

Property Value

int

Message

The error message associated with this exception.

```
string Message { get; }
```

Property Value

string

Type

The category that this exception belongs to.

```
ExceptionType Type { get; }
```

Property Value

ExceptionType

Interface IExceptionManager

Namespace: EdaIntegrationContract.Exceptions

Assembly: EdaIntegration.Contract.dll

The manager of exceptions that occurred during document processing in a case

```
public interface IExceptionManager
```

Properties

Properties

Provides access to the Exception properties for this case.

```
IExceptionProperties Properties { get; }
```

Property Value

IExceptionProperties

The *ExceptionProperties* for accessing exception related properties for this case.

Methods

All(IExceptionFilter, bool)

Retrieves all exceptions that have occurred in a case

```
IEnumerable<IExceptionItem> All(IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

The filters to apply to limit the exceptions returned excludeItems bool

Do not return exceptions that have the 'ExcludeItem' value set to true/1

Returns

IEnumerable<IExceptionItem>

A list of IExceptionItem

Examples

The following example demonstrates retrieving the a list of exceptions for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exceptions;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        string outputFormat = "{0, 6} {1, 6} {2}";
        Console.WriteLine(outputFormat, "Exc Id", "Doc Id", "Message");
        Console.WriteLine(outputFormat, "======", "======",
"========================================");
        foreach (IExceptionItem exception in edaCase.Exceptions.All())
        {
            Console.WriteLine(outputFormat, exception.Id, exception.DocumentId,
exception.Message);
        }
    }
}
/*
This example produces the following results:

Exc Id Doc Id Message
====== ====== ========================================
     1     26 Eligible file contained no content.
     2     16 Eligible file contained no content.
     3     16 File type could not be identified.
     4     16 0 Byte File.
     5    122 File is encrypted.
     6    122 File contains no content due to encryption.
     7    115 File is encrypted.
     8    115 File contains no content due to encryption.
     9    114 File is encrypted.
    10    114 File contains no content due to encryption.
    11     13 File is encrypted.
    12     13 File contains no content due to encryption.
    13     11 Eligible file contained no content.
    14     17 Eligible file contained no content.
    15    116 Eligible file contained no content.
    16    117 Eligible file contained no content.
*/
```

The following example demonstrates retrieving the a list of exceptions for encrypted files and file type detection issues that occurred when processing documents in a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exceptions;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        IExceptionFilter filt = edaCase.Exceptions.NewExceptionFilter();
        filt.Types = new List<ExceptionType>() { ExceptionType.EncryptedFile,
ExceptionType.OcrError };

        string outputFormat = "{0, 6} {1, 6} {2}";
        Console.WriteLine(outputFormat, "Exc Id", "Doc Id", "Message");
        Console.WriteLine(outputFormat, "======", "======",
"========================================");
        foreach (IExceptionItem exception in edaCase.Exceptions.All(filt))
        {
            Console.WriteLine(outputFormat, exception.Id, exception.DocumentId,
exception.Message);
        }
    }
}
/*
This example produces the following results:
```

```
Exc Id Doc Id Message
====== ====== ========================================
     3     16 File type could not be identified.
     5    122 File is encrypted.
     7    115 File is encrypted.
     9    114 File is encrypted.
    11     13 File is encrypted.
*/
```

Any(IExceptionFilter, bool)

Returns whether there are any exceptions (optionally matching filter criteria, and/or excluding exceptions that have ExcludeItem set to 1.)

```
bool Any(IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

The filters to apply to limit the exceptions to look for

`excludeItems` bool

Do not include exceptions that have the 'ExcludeItem' value set to true/1

Returns

bool

true if there are any exceptions, false otherwise

AnyExcludedItems()

Returns whether any exceptions have ExcludeItem set to 1.

```
bool AnyExcludedItems()
```

Returns

bool

true if at least one exception is excluded, false otherwise

ById(int)

Retrieves a specific exception by its unique identifier

```
IExceptionItem ById(int id)
```

Parameters

`id` int

The unique identifier of the custodian to find

Returns

IExceptionItem

The IExceptionItem with the supplied unique identifier

Examples

The following example demonstrates retrieving an exception using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exceptions;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExceptionItem exception = edaCase.Exceptions.ById(3);
        Console.WriteLine("Id: {0},  Doc Id: {1}  Message: {2}", exception.Id,
exception.DocumentId, exception.Message);
    }
}
/*
This example produces the following results:

Id: 3,  Doc Id: 16  Message: File type could not be identified.
 */
```

Exceptions

EdaApiException

Thrown if an exception cannot be found with the specified Id

Checksum(bool)

Gets a checksum representing the list of all exceptions in the case

```
long Checksum(bool excludeItems = false)
```

Parameters

`excludeItems` bool

Do not include exceptions that have the 'ExcludeItem' value set to true/1

Returns

long

Checksum

Count(IExceptionFilter, bool)

Gets the number of exceptions that will be retrieved by the specified filter

```
int Count(IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

The filter to apply

`excludeItems` bool

Do not count exceptions that have the 'ExcludeItem' value set to true/1

Returns

int

Exception count

CountsByType(IExceptionFilter, bool)

Retrieves the count of exceptions for each exception type that meets an optionally supplied set of filter criteria. If no filters are supplied then counts are returned for each exception type for which exceptions occurred.

```
List<KeyValuePair<ExceptionType, int>> CountsByType(IExceptionFilter filter = null, bool
excludeItems = false)
```

Parameters

`filter` IExceptionFilter

The filters to apply to limit the exception values returned

`excludeItems` bool

Do not count exceptions that have the 'ExcludeItem' value set to true/1

Returns

List<KeyValuePair<ExceptionType, int>>

A key/value pair representing the exception type (key) and the number of exceptions that occurred for that type (value).

Examples

The following example demonstrates retrieving the count of exceptions for all exception types.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exceptions;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        Console.WriteLine("Exceptions with counts");
        Console.WriteLine("======================");
        foreach (KeyValuePair<ExceptionType, int> kvp in edaCase.Exceptions.CountsByType())
        {
            Console.WriteLine("{0, -16}  {1, 4}", kvp.Key, kvp.Value);
        }
    }
}
/*
This example produces the following results:

Exceptions with counts
======================
EmptyFile            1
EncryptedFile        4
NoContent           10
UnknownFileType      1
 */
```

DismissErrors(IExceptionFilter)

Sets the [ExcludeItem] column value to 1 for the exceptions specified by the filter.

```
void DismissErrors(IExceptionFilter filter = null)
```

Parameters

`filter` IExceptionFilter

Exception filter

ExcludeItems(IEnumerable<int>)

Sets the [ExcludeItem] column value to 1 for the specified Exception IDs. Using the [Id] column as the key instead of [InventoryId] because there are exceptions, such as InventoryErrors, where InventoryId is null because no document was ever created.

```
void ExcludeItems(IEnumerable<int> ids)
```

Parameters

`ids` IEnumerable<int>

List of IDs of exceptions to set as excluded

FilterValues(ExceptionFilterType, bool)

Retrieves the values and the count of exceptions for a specified filter type.

```
List<IExceptionFilterValue> FilterValues(ExceptionFilterType filterType, bool excludeItems
= false)
```

Parameters

`filterType` ExceptionFilterType

The type of filter to return values for

`excludeItems` bool

Do not retrieve exceptions that have the 'ExcludeItem' value set to true/1

Returns

List<IExceptionFilterValue>

A list of IExceptionFilterValue representing the unique identifier of the value, its description and the count of exceptions for that value

Examples

The following example demonstrates retrieving the count of exceptions for all file types.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exceptions;

class Sample
{
    static void Main()
    {
        /* Connect to the case */
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        /* Set up the output formatting */
        string outputFormat = "{0, 6}  {1, -50} {2, 7}";
        Console.WriteLine(outputFormat, "Exc Id", "Message", "Exc Cnt");
        Console.WriteLine(outputFormat, new string('=', 6), new string('=', 50), new
string('=', 7));

        /* Print each filter value */
        foreach (var exceptionFilterValue in
edaCase.Exceptions.FilterValues(ExceptionFilterType.FileType))
        {
            Console.WriteLine("{0, 6} {1, -50} {2, 4}", exceptionFilterValue.Id,
exceptionFilterValue.Description, exceptionFilterValue.ExceptionCount);
        }
    }
}
/*
This example produces the following results:

Exc Id  FileType                                            Exc Cnt
======  ================================================== =======
   185  JPEG File Interchange Format Image                       4
  2210  MS PowerPoint 2007-2010 Presentation (Open XML)          2
  2208  MS Word 2007-2010 Document (Open XML)                    5
   229  MS Word 97-2003 Document (OLE)                           2
     0  Unidentified                                             3
*/
```

LastExceptionId(IExceptionFilter, bool)

Gets the ID of the last exception, for the specified filter, or 0 if there are no exceptions.

```
int LastExceptionId(IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

The filter to apply

`excludeItems` bool

Do not return exceptions that have the 'ExcludeItem' value set to true/1

Returns

int

ID of the last exception, or 0 if no exception exist

NewExceptionFilter()

Returns an empty exception filter object to use when querying for exception data

```
IExceptionFilter NewExceptionFilter()
```

Returns

IExceptionFilter

An empty exception filter

NextPage(int, int, IExceptionFilter, bool)

Gets the next page of exceptions

```
IEnumerable<IExceptionItem> NextPage(int lastItemId, int pageSize, IExceptionFilter filter =
null, bool excludeItems = false)
```

Parameters

`lastItemId` int

The id of the last exception item, page starts with the next exception

`pageSize` int

The number of exceptions to return

`filter` IExceptionFilter

The filter to apply

`excludeItems` bool

Do not return exceptions that have the 'ExcludeItem' value set to true/1

Returns

IEnumerable<IExceptionItem>

List of exceptions

ReProcess(IEnumerable<IExceptionItem>)

Queues up the documents with the provided list of exceptions to be reprocessed

```
void ReProcess(IEnumerable<IExceptionItem> exceptions)
```

Parameters

`exceptions` IEnumerable<IExceptionItem>

The list of exceptions to reprocess

ReProcessAll(IExceptionFilter, IEnumerable<IExceptionItem>, bool)

Reprocess all exceptions met by the filter criteria

```
void ReProcessAll(IExceptionFilter filter, IEnumerable<IExceptionItem> exceptionsToExclude =
null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

The filters to apply to determine the exceptions to be reprocessed

`exceptionsToExclude` IEnumerable<IExceptionItem>

An optional list of exceptions to exclude from the reprocessing

`excludeItems` bool

Do not reprocess exceptions that have the 'ExcludeItem' value set to true/1

SaveAllToCsv(string, IEnumerable<int>, IExceptionFilter, bool)

Saves all exceptions allowed by filter to csv except specified ids

```
void SaveAllToCsv(string filename, IEnumerable<int> exceptionIdsToSkip = null,
IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filename` string

Name of csv file to create

`exceptionIdsToSkip` IEnumerable<int>

Ids of exceptions to skip

`filter` IExceptionFilter

Filter to apply

`excludeItems` bool

Do not save exceptions that have the 'ExcludeItem' value set to true/1

SaveExcludedToCsv(string, bool)

Save all exceptions where ExcludeItem is 1 to the specified CSV file.

```
void SaveExcludedToCsv(string filename, bool append)
```

Parameters

`filename` string

Name of csv file to create

`append` bool

Append to the file if it exists

SaveSelectedToCsv(string, IEnumerable<int>)

Saves the specified exceptions to a csv file

```
void SaveSelectedToCsv(string filename, IEnumerable<int> exceptionIdsToSave)
```

Parameters

`filename` string

The name of the csv file to create

`exceptionIdsToSave` IEnumerable<int>

A list of ids of exceptions to save

UpdateProperties()

Saves the exception properties to the case.

```
void UpdateProperties()
```

Examples

The following example demonstrates updating the properties for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exceptions;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Turn off conditional exceptions
        edaCase.Exceptions.Properties.ExceptionsForConditionalAnalysis = false;

        // Save the changes to the case
        edaCase.Exceptions.UpdateProperties();
    }
}
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IExceptionProperties

Namespace: EdaIntegrationContract.Exceptions

Assembly: EdaIntegration.Contract.dll

Represents properties that affect the creation of exceptions for the case.

```
public interface IExceptionProperties
```

Properties

ExceptionsForConditionalAnalysis

Enable or disable creation of conditional exceptions such as EmptyFile, NoContent, of UnknownFileType

```
bool ExceptionsForConditionalAnalysis { get; set; }
```

Property Value

bool

See Also

UpdateProperties()

Namespace EdaIntegrationContract.Exports

Interfaces

IExport

The parameters used for exporting processing and the resulting exported documents and metadata.

IExportConfig

The configuration parameters for an export

IExportDocument

Document metadata specific to or derived during the export process

IExportDocumentManager

Manages access to documents exported from a Case

IExportError

An error that occurred attempting to export a document

IExportErrorManager

Manages access to errors that occurred during export processing of a case

IExportManager

Manages the Exports associated with a case.

IExportProperties

Provides access to the Export Properties for this case.

Enums

EmailItemOrderBy

Indicates the sort order for email items in the export

ExportMessageFormat

The available formats that can be used when exporting email messages

ExportScope

The scope of documents to export

ExportSetOrderBy

Fields used to order the creation of the export set

ExportState

The current state of processing an export

ExportType

The type of export to perform.

> **NOTE**
>
> Native and LAW Direct exports are the only types currently supported by the Integration Library.

LoadFileFormat

The format of the load file to generate from the export

Enum EmailItemOrderBy

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

Indicates the sort order for email items in the export

```
public enum EmailItemOrderBy
```

Fields

`SentOnNewestToOldest = 1`

`SentOnOldestToNewest = 0`

Enum ExportMessageFormat

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

The available formats that can be used when exporting email messages

```
public enum ExportMessageFormat
```

Fields

`Html = 1`

Html, images stored externally

`HtmlOrMhtml = 2`

Html or Mhtml depending on images

`Mhtml = 3`

Mime Html storage for all types. Eligibility is the same as Html

`Native = 0`

Native format (msg, eml etc..)

Enum ExportScope

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

The scope of documents to export

```
public enum ExportScope
```

Fields

`Filtered = 1`

The export will include all items that pass all filters

`Tagged = 0`

The export will be generated based off of a set of provided tags

Enum ExportSetOrderBy

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

Fields used to order the creation of the export set

```
public enum ExportSetOrderBy
```

Fields

`Custodian = 0`

`Session = 1`

`Source = 2`

Enum ExportState

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

The current state of processing an export

```
public enum ExportState
```

Fields

`Complete = 2`

Export processing is complete

`ExceptionOccurred = 3`

A fatal error occurred which prevented the processing from completing

`InProgress = 1`

Export is currently being processed

`Pending = 0`

Export is queued up to be processed

Enum ExportType

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

The type of export to perform.

NOTE

Native and LAW Direct exports are the only types currently supported by the Integration Library.

```
public enum ExportType
```

Fields

`CloudNineDirect = 2`

CloudNine Review Export

`LAWDirect = 1`

LAW Direct Export

`Native = 0`

Export native files

Interface IExport

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

The parameters used for exporting processing and the resulting exported documents and metadata.

```
public interface IExport
```

Properties

Config

The configuration properties for the export

```
IExportConfig Config { get; set; }
```

Property Value

IExportConfig

Created

The date and time that the export was created

```
DateTime Created { get; }
```

Property Value

DateTime

Remarks

All date/time values are stored in UTC

Documents

Provides access to the documents that were exported

```
IExportDocumentManager Documents { get; }
```

Property Value

IExportDocumentManager

Errors

Provides access to any errors that were generated during the export processing.

```
IExportErrorManager Errors { get; }
```

Property Value

IExportErrorManager

Exception

An exception that was thrown during the processing of the export which prevented the processing from completing successfully. If no exception occurred then this value will be Null.

```
Exception Exception { get; }
```

Property Value

Exception

Id

The unique identifier for the export

```
int Id { get; }
```

Property Value

int

LastRun

The date and time that this export was last processed.

```
DateTime? LastRun { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

LawSessionId

The LAW Session associated with this export, if it is an LDE or Turbo Import

```
int LawSessionId { get; }
```

Property Value

int

Name

The name of the export.

```
string Name { get; set; }
```

Property Value

string

Remarks

Export names can be up to 50 characters and must be unique to the case.

NumErrors

The number of errors that occurred when exporting documents.

```
int NumErrors { get; }
```

Property Value

int

NumExported

The number of documents that have been successfully exported. This value can be used to determine the progress of the export processing.

```
int NumExported { get; }
```

Property Value

int

NumExtracted

The number of documents that have been extracted from the source data. This value is used for calculating estimated time remaining for Turbo Import ingestion processing.

```
int NumExtracted { get; }
```

Property Value

int

RunCount

The number of times this export ran

```
int RunCount { get; }
```

Property Value

int

State

The current state of the export.

```
ExportState State { get; }
```

Property Value

ExportState

TotalDocuments

The total number of documents to process for the export. This value is not calculated until the export processing is started.

```
int TotalDocuments { get; }
```

Property Value

int

Type

The type of export

```
ExportType Type { get; }
```

Property Value

ExportType

Interface IExportConfig

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

The configuration parameters for an export

```
public interface IExportConfig
```

Properties

AlphaNumericNumbering

Whether the NumberingSeed is an alphanumeric number format

```
bool AlphaNumericNumbering { get; set; }
```

Property Value

bool

EmailSortOrder

Indicates the sort order for email items in the sort

```
EmailItemOrderBy EmailSortOrder { get; set; }
```

Property Value

EmailItemOrderBy

ExportDirectory

The root directory where exported documents and load files will be placed. This option is only applicable if including native files (IncludeNativeFiles),

```
string ExportDirectory { get; set; }
```

Property Value

string

ExportExtractionFailures

Whether native file extraction failures should be exported with place-holder .err files.

```
bool ExportExtractionFailures { get; set; }
```

Property Value

bool

ExportSetOrderBys

Fields used to order the creation of the export set

```
IEnumerable<ExportSetOrderBy> ExportSetOrderBys { get; set; }
```

Property Value

IEnumerable<ExportSetOrderBy>

ExportTextToFile

Whether the text extracted from the documents is exported to text files

```
bool ExportTextToFile { get; set; }
```

Property Value

bool

GeneratedLoadFileFormat

Identifies the load file format to generate during the export.

```
LoadFileFormat GeneratedLoadFileFormat { get; set; }
```

Property Value

LoadFileFormat

IncludeEmailsInThread

Whether emails in threads are included as a part of the export

```
bool IncludeEmailsInThread { get; set; }
```

Property Value

bool

IncludeNativeFiles

Whether native files are copied to the export directory as a part of the export processing

```
bool IncludeNativeFiles { get; set; }
```

Property Value

bool

LawCaseIni

Path to the LAW ini file for the case

```
string LawCaseIni { get; set; }
```

Property Value

string

LawCaseName

THe LAW case name

```
string LawCaseName { get; set; }
```

Property Value

string

MessageFormat

The preferred format when exporting email messages. This option only applies when exporting native files (see IncludeNativeFiles)

```
ExportMessageFormat MessageFormat { get; set; }
```

Property Value

ExportMessageFormat

NumberAttachments

Whether attachments should be identified separately in the export numbering scheme

```
bool NumberAttachments { get; set; }
```

Property Value

bool

NumberingSeed

The seed value to use when adding new items to the export

```
string NumberingSeed { get; set; }
```

Property Value

string

Scope

The scope of the export (all filtered documents or documents tagged with certain tags)

```
ExportScope Scope { get; set; }
```

Property Value

ExportScope

TagsToExclude

For an ExportScope of "Tagged", documents tagged with these tags will be excluded from the export.

```
IEnumerable<ITag> TagsToExclude { get; set; }
```

Property Value

IEnumerable<ITag>

TagsToInclude

For an ExportScope of "Tagged", documents tagged with these tags will be included in the export unless they have also been tagged with a tag that appears in TagsToExclude.

```
IEnumerable<ITag> TagsToInclude { get; set; }
```

Property Value

IEnumerable<ITag>

UseLegacyFolderStructure

Identifies whether to create the folder structure using pre-7.2 defaults

```
bool UseLegacyFolderStructure { get; set; }
```

Property Value

bool

Interface IExportDocument

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

Document metadata specific to or derived during the export process

```
public interface IExportDocument : IDocument
```

Inherited Members

IDocument.CompositeName , IDocument.CompressedSize , IDocument.ContainerPath , IDocument.Custodian , IDocument.DateFilterLowerBound , IDocument.DateFilterUpperBound , IDocument.Decrypted , IDocument.Directory , IDocument.DocExtension , IDocument.DocumentType , IDocument.EDocMetadata , IDocument.EmailMetadata , IDocument.Encrypted , IDocument.FamilyParentId , IDocument.FileName , IDocument.FileType , IDocument.FileTypeDescription , IDocument.FileUri , IDocument.HasAttachments , IDocument.Hash , IDocument.Id , IDocument.Imported , IDocument.ImportSet , IDocument.InventoryId , IDocument.IsArchiveItem , IDocument.IsAttachment , IDocument.IsCompound , IDocument.IsCompoundChild , IDocument.DedupeStatus , IDocument.IsEmailAttachment , IDocument.IsSuspectExtension , IDocument.IsTextStoredOnDisk , IDocument.Level , IDocument.MimeType , IDocument.ParentId , IDocument.Password , IDocument.Size , IDocument.Source , IDocument.SourceDocumentId , IDocument.SourceFile , IDocument.Text , IDocument.TextLocation , IDocument.TextSize

Properties

AttachmentMetadata

Information relating to document attachments. If there is no attachment data this will be Null.

```
IAttachmentMetadata AttachmentMetadata { get; }
```

Property Value

IAttachmentMetadata

DuplicateMetadata

Information relating to Deduplication analysis. If there is no duplicate data, or if the deduplication process has been disabled, then this will be Null.

```
IDupeMetadata DuplicateMetadata { get; }
```

Property Value

IDupeMetadata

EmailThreadingMetadata

Email threading information. If there is no email threading data, or if the Email Threading process has been disabled for Analysis, then this will be Null.

```
IEmailThreadMetadata EmailThreadingMetadata { get; }
```

Property Value

IEmailThreadMetadata

ExportNativeFileSize

The size of the exported native file in bytes. 0 if native file extraction fails or if the document was not exported.

```
long ExportNativeFileSize { get; }
```

Property Value

long

ExportNativeFileTitle

The name of the native file that was exported, if IncludeNativeFiles was selected in the export configuration. The exported file extension is based on DocExtension and the email message format that was selected in the export configuration.

```
string ExportNativeFileTitle { get; }
```

Property Value

string

ExportNumber

Based on the NumberingSeed value specified, this value increments by one for each document in the export.

```
string ExportNumber { get; }
```

Property Value

string

ExportPath

The sub-directory below the ExportDirectory where the native file or text file was placed.

```
string ExportPath { get; }
```

Property Value

string

ExportTextFileTitle

The name of the text file that was exported, if ExportTextToFile was selected in the export configuration.

```
string ExportTextFileTitle { get; }
```

Property Value

string

Languages

Languages identified in this document.

```
IEnumerable<string> Languages { get; }
```

Property Value

IEnumerable<string>

NearDuplicateMetadata

Information relating to Near Duplicate analysis. If there is no near duplicate data, or if the Near Duplicate process has been disabled for Analysis, then this will be Null.

```
INearDupeMetadata NearDuplicateMetadata { get; }
```

Property Value

INearDupeMetadata

Tags

The list of tags that have been applied to this document.

```
IEnumerable<ITag> Tags { get; set; }
```

Property Value

IEnumerable<ITag>

Interface IExportDocumentManager

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

Manages access to documents exported from a Case

```
public interface IExportDocumentManager
```

Methods

All(bool)

Retrieves all documents for an export

```
IEnumerable<IExportDocument> All(bool includeReceivedDocs = false)
```

Parameters

`includeReceivedDocs` bool

Returns

IEnumerable<IExportDocument>

A list of IExportDocuments that have been exported

Examples

The following example demonstrates retrieving the list of documents for an export and printing each name to the system console.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Get the export
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");

        foreach (IExportDocument doc in export.Documents.All())
        {
            Console.WriteLine("{0}:  {1}", doc.ExportNumber, doc.FileName);
        }
    }
}
/*
This example produces the following results:

00001160:  Unprofitable actions.msg
00001155:  Four Attributes.msg
00002241:  Technology outlook.pdf
00001163:  Load Forecasting Technology.msg
00002240:  Book1.xlsx
00001159:  Monthly data.msg
00001170:  Viewpoint from above.msg
00001168:  Scope.docx
00001166:  Key Stats YTD 2011.xlsx
00001167:  Forecast 2012.pptx
 */
```

MarkAsNotReceived(IEnumerable<int>)

Clears the received/processed state for a set of documents within an export. After calling this, the documents will appear again in the list returned from All(bool)

```
void MarkAsNotReceived(IEnumerable<int> documentIds)
```

Parameters

documentIds IEnumerable<int>

The list of documents ids to update

Examples

The following example demonstrates retrieving the first 500 documents from an export and clearing their received/processed state.

```csharp
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Get the export
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");

        // Retrieve the list of documents that were exported
        IExportDocumentManager mgr = export.Documents;

        // Retrieve the identifiers of the first 500 documents
        List<int> docIds = mgr.All(true).Take(500).Select(x => x.InventoryId).ToList();

        // Clear their received state
        mgr.MarkAsNotReceived(docIds);
    }
}
```

Remarks

This method supports updating up to 1000 documents at a time. Any list longer than 1000 will cause an exception to be thrown.

MarkAsNotReceived(int)

Clears the received/processed state for a document within an export. After calling this, the document will appear again in the list returned from All(bool)

```csharp
void MarkAsNotReceived(int documentId)
```

Parameters

documentId int

The unique identifier of the document to update

Examples

The following example demonstrates marking a document as not having been received and processed. Doing this ensures that subsequent calls to All(bool) will include this document.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Get the export
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");

        // Retrieve the list of documents that were exported
        IExportDocumentManager mgr = export.Documents;

        // Loop through each document and clear it's received state
        foreach (IExportDocument doc in mgr.All(true))
        {
            Console.WriteLine("{0}:  {1}", doc.ExportNumber, doc.FileName);

            // Cleat the received state of the document
            mgr.MarkAsNotReceived(doc.InventoryId);
        }
    }
}
/*
This example produces the following results:

00001160:  Unprofitable actions.msg
00001155:  Four Attributes.msg
00002241:  Technology outlook.pdf
00001163:  Load Forecasting Technology.msg
00002240:  Book1.xlsx
00001159:  Monthly data.msg
```

```
00001170:  Viewpoint from above.msg
00001168:  Scope.docx
00001166:  Key Stats YTD 2011.xlsx
00001167:  Forecast 2012.pptx
 */
```

Remarks

This method exists to aid in prototyping or testing using the EdaIntegration Library.

MarkAsReceived(IEnumerable<int>)

Marks the documents as having been received/processed

```
void MarkAsReceived(IEnumerable<int> documentIds)
```

Parameters

documentIds IEnumerable<int>

The list of documents ids to mark as received

Examples

The following example demonstrates retrieving the list of documents for an export and printing each name to the system console. Additionally the document ids are accumulated and updated to show that the documents have been received and processed. This ensures that subsequent calls to All(bool) return only "new" documents.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Get the export
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");
        // Create a list to hold the ids of the documents that have been received/processed
        List<int> docsReceived = new List<int>();

        // Set the number of documents to process before we update their received state
        const int batchSize = 10;

        // Retrieve the list of documents that were exported
        ExportDocumentManager mgr = export.Documents;

        // Loop through each document
        foreach (IExportDocument doc in mgr.All())
        {
            Console.WriteLine("{0}:  {1}", doc.ExportNumber, doc.FileName);

            // Accumulate the document ids so we can mark them as received.
            docsReceived.Add(doc.InventoryId);
            if (docsReceived.Count >= batchSize)
            {
                // Mark items as received
                mgr.MarkAsReceived(docsReceived);

                // Clear the received list
                docsReceived.Clear();
            }
        }
    }
}
/*
This example produces the following results:

00001160:  Unprofitable actions.msg
00001155:  Four Attributes.msg
00002241:  Technology outlook.pdf
00001163:  Load Forecasting Technology.msg
00002240:  Book1.xlsx
00001159:  Monthly data.msg
00001170:  Viewpoint from above.msg
00001168:  Scope.docx
00001166:  Key Stats YTD 2011.xlsx
00001167:  Forecast 2012.pptx
 */
```

Remarks

This method supports updating up to 1000 documents at a time. Any list longer than 1000 will cause an exception to be thrown.

MarkAsReceived(int)

Marks the document as having been received/processed

```
void MarkAsReceived(int documentId)
```

Parameters

`documentId` int

The unique identifier of the document to mark as received

Examples

The following example demonstrates marking documents in as having been received and processed. Doing so ensures that subsequent calls to All(bool) return only "new" documents. The example retrieves the list of documents for an export, prints each name to the system console, and then marks them as received so that they will not be returned in subsequent calls.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Get the export
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");

        // Retrieve the list of documents that were exported
        IExportDocumentManager mgr = export.Documents;

        // Loop through each document
        foreach (IExportDocument doc in mgr.All())
        {
            Console.WriteLine("{0}:  {1}", doc.ExportNumber, doc.FileName);

            // Mark the document as received
            mgr.MarkAsReceived(doc.InventoryId);
        }
    }
}
/*
This example produces the following results:

00001160:  Unprofitable actions.msg
00001155:  Four Attributes.msg
00002241:  Technology outlook.pdf
00001163:  Load Forecasting Technology.msg
00002240:  Book1.xlsx
00001159:  Monthly data.msg
00001170:  Viewpoint from above.msg
00001168:  Scope.docx
00001166:  Key Stats YTD 2011.xlsx
00001167:  Forecast 2012.pptx
 */
```

Interface IExportError

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

An error that occurred attempting to export a document

```
public interface IExportError : IEdaIntegrationUniqueEntity, IEdaIntegrationEntity
```

Properties

Created

The date/time the error was created

```
DateTime Created { get; }
```

Property Value

DateTime

Detail

Detailed information for the error

```
string Detail { get; }
```

Property Value

string

DocumentId

The unique identifier of the document to which the error belongs

```
int DocumentId { get; }
```

Property Value

int

ExportId

The id of the export in which the error occurred

```
int ExportId { get; }
```

Property Value

int

FileReference

The path to the file in which the error occurred

```
string FileReference { get; }
```

Property Value

string

Id

A unique identifier for the error

```
int Id { get; }
```

Property Value

int

Message

The error message

```
string Message { get; }
```

Property Value

string

Interface IExportErrorManager

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

Manages access to errors that occurred during export processing of a case

```
public interface IExportErrorManager
```

Methods

All(IExceptionFilter, bool)

Retrieves the list of errors that occurred when exporting documents for this export.

```
IEnumerable<IExportError> All(IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

An optional filter

`excludeItems` bool

Do not return export errors that have the 'ExcludeItem' value set to true/1

Returns

IEnumerable<IExportError>

A list of IExportErrors. If no errors exist then an empty list is returned.

Examples

The following example demonstrates retrieving the errors for an export.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Get the export
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");

        foreach (IExportError error in export.Errors.All())
        {
            Console.WriteLine("Id: {0}", error.Id);
            Console.WriteLine("Message: {0}", error.Message);
            Console.WriteLine("Doc Id: {0}", error.DocumentId);
            Console.WriteLine("");
        }
    }
}
/*
This example produces the following results:

Id: 3
Message: Source file not found. \\NetworkDrive\Folder1\Folder2\Documents\samples.pst
Doc Id: 24

Id: 4
Message: Source file not found. \\NetworkDrive\Folder1\Folder2\Documents\samples.pst
Doc Id: 25
 */
```

Any(IExceptionFilter, bool)

Returns whether there are any export errors for the selected export.

```
bool Any(IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

An optional filter

excludeItems bool

Do not return export errors that have the 'ExcludeItem' value set to true/1

Returns

bool

true if there are any errors for the export

AnyExcludedItems()

Returns whether any export errors have ExcludeItem set to 1.

```
bool AnyExcludedItems()
```

Returns

bool

true if at least one export error is excluded, false otherwise

ById(int)

Retrieves a specific error for an export.

```
IExportError ById(int exportErrorId)
```

Parameters

exportErrorId int

The unique identifier of the export error to retrieve.

Returns

IExportError

The requested IExportError

Examples

The following example demonstrates retrieving a specific error for an export.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();

        // Get the export
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");

        IExportError error = export.Errors.ById(4);
        Console.WriteLine("Id: {0}", error.Id);
        Console.WriteLine("Message: {0}", error.Message);
        Console.WriteLine("Doc Id: {0}", error.DocumentId);
    }
}
/*
This example produces the following results:

Id: 4
Message: Source file not found. \\NetworkDrive\Folder1\Folder2\Documents\samples.pst
Doc Id: 25
 */
```

Exceptions

EdaApiException

Thrown if the error does not exist

Checksum(bool)

Gets a checksum for the current set of export errors

```
long Checksum(bool excludeItems = false)
```

Parameters

`excludeItems` bool

Do not include export errors that have the 'ExcludeItem' value set to true/1

Returns

long

Checksum

Count(IExceptionFilter, bool)

Gets the number of exceptions filtered by the supplied filter

```
int Count(IExceptionFilter filter = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

Optional exception filter

`excludeItems` bool

Do not count export errors that have the 'ExcludeItem' value set to true/1

Returns

int

Number of exceptions

DismissErrors(IExceptionFilter)

Sets the [ExcludeItem] column value to 1 for the exceptions specified by the filter.

```
void DismissErrors(IExceptionFilter filter = null)
```

Parameters

`filter` IExceptionFilter

Exception filter

ExcludeItems(IEnumerable<int>)

Sets the [ExcludeItem] column value to 1 for the specified InventoryItem IDs.

```
void ExcludeItems(IEnumerable<int> inventoryIds)
```

Parameters

`inventoryIds` IEnumerable<int>

List of IDs of export errors to set as excluded

FilterValues(ExceptionFilterType, bool)

Gets the possible values for an exception filter

```
IEnumerable<IExceptionFilterValue> FilterValues(ExceptionFilterType filterType, bool
excludeItems = false)
```

Parameters

`filterType` ExceptionFilterType

the type of filter for which to get values

`excludeItems` bool

Do not retrieve export errors that have the 'ExcludeItem' value set to true/1

Returns

IEnumerable<IExceptionFilterValue>

List of possible exception filter values

LastExportErrorId(bool)

Gets the ID of the last export error for the specified Export, if any exist.

```
int LastExportErrorId(bool excludeItems = false)
```

Parameters

`excludeItems` bool

Do not include export errors that have the 'ExcludeItem' value set to true/1

Returns

int

The ID of the last export error, or 0 if there are no errors

NextPage(int, int, IExceptionFilter, bool)

Gets the next page of exceptions filtered by supplied filter

```
IEnumerable<IExportError> NextPage(int lastItemId, int pageSize, IExceptionFilter filter =
null, bool excludeItems = false)
```

Parameters

`lastItemId` int

The id of the most recently retrieved exception

`pageSize` int

The number of exception to retrieve

`filter` IExceptionFilter

An optional filter

`excludeItems` bool

Do not return export errors that have the 'ExcludeItem' value set to true/1

Returns

IEnumerable<IExportError>

List of exceptions

ReTry(IEnumerable<IExportError>)

Clears the error for the provided documents and then queues up the Export process. NOTE: There is an implicit assumption that all export errors to retry come from the same export.

```
void ReTry(IEnumerable<IExportError> exportErrors)
```

Parameters

`exportErrors` IEnumerable<IExportError>

The list of export errors to reprocess

ReTryAll(IExceptionFilter, IEnumerable<IExportError>, bool)

Reprocess all export errors met by the filter criteria

```
void ReTryAll(IExceptionFilter filter = null, IEnumerable<IExportError>
exportErrorsToExclude = null, bool excludeItems = false)
```

Parameters

`filter` IExceptionFilter

The filters to apply to determine the export errors to be reprocessed

`exportErrorsToExclude` IEnumerable<IExportError>

An optional list of export errors to exclude from the reprocessing

`excludeItems` bool

Do not retry export errors that have the 'ExcludeItem' value set to true/1

SaveAllToCsv(string, IExceptionFilter, IEnumerable<int>, bool)

Saves all exceptions (filtered) to csv file, skipping specified exceptions

```
void SaveAllToCsv(string filename, IExceptionFilter filter, IEnumerable<int> errorIdsToSkip
= null, bool excludeItems = false)
```

Parameters

`filename` string

The name of the csv file to create

`filter` IExceptionFilter

An optional filter

`errorIdsToSkip` IEnumerable<int>

Optional list ids of errors to skip

`excludeItems` bool

Do not save export errors that have the 'ExcludeItem' value set to true/1

SaveExcludedToCsv(string, bool)

Save all export errors where ExcludeItem is 1 to the specified CSV file.

```
void SaveExcludedToCsv(string filename, bool append)
```

Parameters

`filename` string

Name of csv file to create

`append` bool

Append to the file if it exists

SaveSelectedToCsv(string, IEnumerable<int>)

Saves exceptions with supplied ids to csv file

```
void SaveSelectedToCsv(string filename, IEnumerable<int> errorIdsToSave)
```

Parameters

`filename` string

The name of the csv file to create

`errorIdsToSave` IEnumerable<int>

ids of exceptions to save

Interface IExportManager

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

Manages the Exports associated with a case.

```
public interface IExportManager
```

Properties

ActiveExport

Retrieve active export in the case

```
IExport ActiveExport { get; }
```

Property Value

IExport

Properties

Provides access to the Export properties for this case.

```
IExportProperties Properties { get; }
```

Property Value

IExportProperties

Methods

All()

Retrieves the list of Exports for a case.

```
IEnumerable<IExport> All()
```

Returns

IEnumerable<IExport>

A list of IExports. If no exports exist for a case then an empty list is returned.

Examples

The following example demonstrates retrieving the a list of exports for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        foreach (IExport export in edaCase.Exports.All())
        {
            Console.WriteLine("Name: {0}", export.Name);
            Console.WriteLine("Type: {0}", export.Type);
            Console.WriteLine("Scope: {0}", export.Config.Scope);
            Console.WriteLine("Last Run: {0}", export.LastRun);
            Console.WriteLine("Total Documents: {0}", export.TotalDocuments);
            Console.WriteLine("Error Count: {0}", export.NumErrors);
            Console.WriteLine("");
        }
    }
}
/*
This example produces the following results:

Name: Export-001
Type: Native
Scope: Filtered
Last Run: 7/24/2014 6:30:59 PM
```

```
Total Documents: 21049
Error Count: 2

Name: Export-002
Type: Native
Scope: Tagged
Last Run: 7/24/2014 10:08:14 PM
Total Documents: 1656
Error Count: 0
```

ById(int)

Retrieves an export using its unique identifier.

```
IExport ById(int exportId)
```

Parameters

`exportId` int

The unique identifier of the export to retrieve.

Returns

IExport

The export with the supplied unique identifier.

Examples

The following example demonstrates retrieving an export using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
```

```
            IExport export = edaCase.Exports.ById(3);
            Console.WriteLine("Name: {0}", export.Name);
            Console.WriteLine("Type: {0}", export.Type);
            Console.WriteLine("Scope: {0}", export.Config.Scope);
            Console.WriteLine("Last Run: {0}", export.LastRun);
            Console.WriteLine("Total Documents: {0}", export.TotalDocuments);
            Console.WriteLine("Error Count: {0}", export.NumErrors);
        }
    }
    /*
    This example produces the following results:

    Name: Export-001
    Type: Native
    Scope: Filtered
    Last Run: 7/24/2014 6:30:59 PM
    Total Documents: 21049
    Error Count: 2
     */
```

Exceptions

EdaApiException

Thrown if an export cannot be found with the specified Id.

ByName(string)

Retrieves an export using its name

```
IExport ByName(string exportName)
```

Parameters

`exportName` string

The name of the export to retrieve

Returns

IExport

The export with the supplied unique identifier

Examples

The following example demonstrates retrieving an export using its name.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");
        Console.WriteLine("Name: {0}", export.Name);
        Console.WriteLine("Type: {0}", export.Type);
        Console.WriteLine("Scope: {0}", export.Config.Scope);
        Console.WriteLine("Last Run: {0}", export.LastRun);
        Console.WriteLine("Total Documents: {0}", export.TotalDocuments);
        Console.WriteLine("Error Count: {0}", export.NumErrors);
    }
}
/*
This example produces the following results:

Name: Export-001
Type: Native
Scope: Filtered
Last Run: 7/24/2014 6:30:59 PM
Total Documents: 21049
Error Count: 2
 */
```

Exceptions

EdaApiException

Thrown if an export cannot be found with the specified name

Create(string, ExportType, string)

Creates a new export for a case and set it as the active export

```
IExport Create(string exportName, ExportType exportType, string baseDirectory = null)
```

Parameters

`exportName` string

The name of the new export

`exportType` ExportType

The type of the new export

`baseDirectory` string

Optional base directory to use for staging LAW Direct export files

Returns

IExport

The export object added to the case

Examples

This example demonstrates how to create a new export for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.Create("Export-001");

        Console.WriteLine("Id: {0}", export.Id);
        Console.WriteLine("Name: {0}", export.Name);
        Console.WriteLine("Type: {0}", export.Type);
        Console.WriteLine("Scope: {0}", export.Config.Scope);
        Console.WriteLine("Last Run: {0}", export.LastRun);
        Console.WriteLine("Total Documents: {0}", export.TotalDocuments);
        Console.WriteLine("Error Count: {0}", export.NumErrors);
    }
}
/*
This example produces the following results:

Id: 3
Name: Export-001
Type: Native
Scope: Filtered
Last Run:
Total Documents: 0
Error Count: 0
 */
```

Delete(int)

Delete an export.

```
void Delete(int exportId)
```

Parameters

exportId int

The unique identifier of the export to delete

Examples

The following example demonstrates how to delete an export

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Select the export to delete
        IExport export = edaCase.Exports.ByName("Export-001");

        // Process the delete
        edaCase.Exports.Delete(export.Id);
    }
}
```

Exceptions

EdaApiException

Thrown if an export with the provided id does not exist

Resume(IExport)

Resume an export if it was cancelled

```
void Resume(IExport export)
```

Parameters

`export` IExport

export to resume

Start(int)

Starts the processing to export the designated documents

```
void Start(int exportId)
```

Parameters

`exportId` int

The unique identifier of the export to process

Examples

The following example demonstrates how to start the export processing for a case

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        IExport export = edaCase.Exports.ByName("Export-001");
        edaCase.Exports.Start(exportId);
    }
}
```

Exceptions

EdaApiException

Thrown if an export with the provided id does not exist

Update(IExport)

Updates the properties for an export

```
void Update(IExport export)
```

Parameters

`export` IExport

The export containing updated data to save

Examples

This example demonstrates updating the properties of an export.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Retrieve an existing export
        IExport export = edaCase.Exports.ByName("Export-001");
        Console.WriteLine("Name: {0}", export.Name);
        Console.WriteLine("Next Number: {0}", export.Config.NumberingSeed);
        Console.WriteLine("Use Alpha Numbering?: {0}", export.Config.Scope);

        // Change its document numbering properties
        export.Config.AlphaNumericNumbering = true;
        export.Config.NumberingSeed = "100a";
        edaCase.Exports.Update(export);

        // Re-retrieve the export and verify the property changes
        export = edaCase.Exports.ByName("Export-001");
        Console.WriteLine("Name: {0}", export.Name);
        Console.WriteLine("Next Number: {0}", export.Config.NumberingSeed);
        Console.WriteLine("Use Alpha Numbering?: {0}", export.Config.Scope);
    }
}
/*
This example produces the following results:

Name: Export-001
Next Number: 00000001
Use Alpha Numbering: False

Name: Export-001
Next Number: 100a
Use Alpha Numbering: True
 */
```

Exceptions

EdaApiException

Thrown if an export with the provided id does not exist

ArgumentException

Thrown if the export configuration parameters are invalid

UpdateProperties()

Saves the export properties to the case.

```
void UpdateProperties()
```

Examples

The following example demonstrates updating the Export properties for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Exports;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        edaCase.Exports.Properties.MaxExportAgents = 4;
        edaCase.Exports.Properties.TimeZoneId = System.TimeZoneInfo.Utc.Id;
        edaCase.Exports.UpdateProperties();
    }
}
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IExportProperties

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

Provides access to the Export Properties for this case.

```
public interface IExportProperties
```

Properties

MaxExportAgents

The maximum number of export agents allowed to run with this case.

```
int MaxExportAgents { get; set; }
```

Property Value

int

Examples

This example demonstrates how to set and retrieve the MaxExportAgents.

```
using Law.EdaIntegration;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the MaxExportAgents value
        edaCase.Exports.Properties.MaxExportAgents = 4;

        // Save the changes to the case
        edaCase.Exports.UpdateProperties();
        // Get the value
        Console.WriteLine("MaxExportAgents: {0}",
edaCase.Exports.Properties.MaxExportAgents);
    }
}
/*
This example produces the following results:

MaxExportAgents: 4
 */
```

Remarks

A value of 0 represents an unlimited number of agents.

Exceptions

EdaApiException

An exception is thrown if the max export agents value is not between 0 and 999.

TimeZoneId

The Id of the selected timezone for the case.

```
string TimeZoneId { get; set; }
```

Property Value

string

Examples

This example demonstrates how to set and retrieve the Id of a time zone item.

```
using Law.EdaIntegration;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the Id of the UTC timezone
        edaCase.Exports.Properties.TimeZoneId = System.TimeZoneInfo.Utc.Id;

        // Save the changes to the case
        edaCase.Exports.UpdateProperties();

        // Get the time zone name from the current Id
        string timeZoneId = edaCase.Exports.Properties.TimeZoneId;

        // Output the standard name of the time zone
        Console.WriteLine("Time zone: {0}",
System.TimeZoneInfo.FindSystemTimeZoneById(timeZoneId).StandardName);
    }
}
/*
This example produces the following results:

Time zone: UTC
*/
```

Remarks

The default value is the Id of UTC time.

Exceptions

EdaApiException

An exception is thrown if the time zone id is not valid.

See Also

UpdateProperties()

Enum LoadFileFormat

Namespace: EdaIntegrationContract.Exports

Assembly: EdaIntegration.Contract.dll

The format of the load file to generate from the export

```
public enum LoadFileFormat
```

Fields

`DAT = 2`

Generate a DAT load file for importing into CloudNine® Concordance® or other applications that support a DAT load file

`EDRM = 1`

Generate an EDRM XML version 1.0 load file

`None = 0`

Do not generate a load file

Namespace EdaIntegrationContract.Filters

Interfaces

IDateRangeFilterDefinition

Defines a date range filter

IDateRangeFilterManager

Manages date range filters for the case

IDeduplicationManager

Manager of all interactions with document deduplication in a case

IDeduplicationProperties

Represents the Deduplication properties for a case.

IFileTypeFilterManager

Manage all file type filtering interactions in a case.

IFileTypeFilterProperties

File type filtering properties for a case.

INistManager

Manager of all interactions with document Nist in a case

INistProperties

Represents the Nist properties for a case.

Enums

DeduplicationMethod

Hashing method used to identify duplicate documents

DeduplicationStatus

Status of documents after running the deduplication process

Enum DeduplicationMethod

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Hashing method used to identify duplicate documents

```
public enum DeduplicationMethod
```

Fields

`Md5 = 0`

Use the MD5 hash value of the document to identify duplicates

`Md5Custodian = 2`

Use the MD5 hash value of the document + custodian Id to identify duplicates by custodian

`Sha1 = 1`

Use the SHA-1 hash value of the document to identify duplicates

`Sha1Custodian = 3`

Use the SHA-1 hash value of the document + custodian Id to identify duplicates by custodian

Enum DeduplicationStatus

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Status of documents after running the deduplication process

```
public enum DeduplicationStatus
```

Fields

`ChildDuplicate = 2`

Document was identified as a child duplicate

`NotDeduped = 0`

Deduplication did not run yet or deduplication is disabled for this case

`NotDuplicate = 3`

Document is not a duplicate of any other documents

`PrimaryDuplicate = 1`

Document was identified as the primary/parent duplicate

Interface IDateRangeFilterDefinition

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Defines a date range filter

```
public interface IDateRangeFilterDefinition
```

Properties

EndDate

Inclusive ending date for the filter

```
DateTime? EndDate { get; set; }
```

Property Value

DateTime?

ID

ID of filter definition (generated)

```
int ID { get; set; }
```

Property Value

int

IncludeItems

Flag whether to include the items selected by this filter

```
bool? IncludeItems { get; set; }
```

Property Value

bool?

StartDate

Inclusive starting date for the filter

```
DateTime? StartDate { get; set; }
```

Property Value

DateTime?

Interface IDateRangeFilterManager

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Manages date range filters for the case

```
public interface IDateRangeFilterManager
```

Methods

AddDateRangeFilter(DateTime, DateTime, bool, bool)

Adds a date range filter

```
IDateRangeFilterDefinition AddDateRangeFilter(DateTime startDate, DateTime endDate, bool
includeItems = true, bool runResolveFilters = true)
```

Parameters

`startDate` DateTime

The beginning date of the date range

`endDate` DateTime

The end date of the range

`includeItems` bool

Flag indicating whether items in the range are included or excluded

`runResolveFilters` bool

Flag indicating whether the system should immediately resolve the results of adding this filter

Returns

IDateRangeFilterDefinition

The date range filter that was just created

DeleteAllDateRangeFilters()

Delete all of the date range filters in the case

```
void DeleteAllDateRangeFilters()
```

DeleteDateRangeFilter(IDateRangeFilterDefinition)

Deletes a date range filter

```
void DeleteDateRangeFilter(IDateRangeFilterDefinition filterDefinition)
```

Parameters

`filterDefinition` IDateRangeFilterDefinition

The data range filter to delete

DeleteDateRangeFilters(IEnumerable<IDateRangeFilterDefinition>)

Deletes multiple date range filters

```
void DeleteDateRangeFilters(IEnumerable<IDateRangeFilterDefinition> filterDefinitions)
```

Parameters

`filterDefinitions` IEnumerable<IDateRangeFilterDefinition>

The list of date range filters to delete

GetAllDateRangeFilters()

Gets all of the date range filters for this case

```
IEnumerable<IDateRangeFilterDefinition> GetAllDateRangeFilters()
```

Returns

IEnumerable<IDateRangeFilterDefinition>

The list of all date range filters currently established for the case

Interface IDeduplicationManager

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Manager of all interactions with document deduplication in a case

```
public interface IDeduplicationManager
```

Properties

Properties

Provides access to the Deduplication properties for this case.

```
IDeduplicationProperties Properties { get; }
```

Property Value

IDeduplicationProperties

The *DeduplicationProperties* for accessing analysis related properties for this case.

Methods

UpdateProperties()

Saves the Deduplication properties to the case.

```
void UpdateProperties()
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IDeduplicationProperties

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Represents the Deduplication properties for a case.

```
public interface IDeduplicationProperties
```

Properties

DeduplicationEnabled

Controls whether or not to run the deduplication process

```
bool DeduplicationEnabled { get; set; }
```

Property Value

bool

DeduplicationMode

Deduplication key that will be used to determine the document duplicate state.

```
DeduplicationMethod DeduplicationMode { get; set; }
```

Property Value

DeduplicationMethod

DuplicatePrioritizationEnabled

If set, deduplication keys are compared against all other keys that have the same custodian value.

```
bool DuplicatePrioritizationEnabled { get; set; }
```

Property Value

bool

IncludeDuplicates

If set, duplicate documents are added to the case normally, including the native files and all associated duplicate fields. Otherwise duplicate documents are completely excluded from the case.

```
bool IncludeDuplicates { get; set; }
```

Property Value

bool

See Also

UpdateProperties()

Interface IFileTypeFilterManager

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Manage all file type filtering interactions in a case.

```
public interface IFileTypeFilterManager
```

Properties

Properties

Provides access to the FileType filtering properties for this case.

```
IFileTypeFilterProperties Properties { get; }
```

Property Value

IFileTypeFilterProperties

The *FileTypeFilterProperties* for accessing filtering related properties for this case.

Methods

IsFileTypeManagementConfigured()

Check if FileType is configured

```
bool IsFileTypeManagementConfigured()
```

Returns

bool

True if file type is configured

UpdateProperties()

Saves the file type filtering properties to the case.

```
void UpdateProperties()
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IFileTypeFilterProperties

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

File type filtering properties for a case.

```
public interface IFileTypeFilterProperties
```

Properties

FileTypeFilteringEnabled

Enables or disables file type filtering.

```
bool FileTypeFilteringEnabled { get; set; }
```

Property Value

bool

IncludeUnspecifiedFileTypes

If true, include any file types that are not specifically configured to be included or excluded. If false, exclude any unconfigured file types.

```
bool IncludeUnspecifiedFileTypes { get; set; }
```

Property Value

bool

Interface INistManager

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Manager of all interactions with document Nist in a case

```
public interface INistManager
```

Properties

Properties

Provides access to the Nist properties for this case.

```
INistProperties Properties { get; }
```

Property Value

INistProperties

The *NistProperties* for accessing filtering related properties for this case.

Methods

IsNistConfigured()

Check if NIST is configured and NIST data is loaded

```
bool IsNistConfigured()
```

Returns

bool

True if NIST has been configured

UpdateProperties()

Saves the Nist properties to the case.

```
void UpdateProperties()
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface INistProperties

Namespace: EdaIntegrationContract.Filters

Assembly: EdaIntegration.Contract.dll

Represents the Nist properties for a case.

```
public interface INistProperties
```

Properties

IncludeItems

If set, Nisted documents are added to the case normally, including the native files and all associated fields. Otherwise Nisted documents are completely excluded from the case.

```
bool IncludeItems { get; set; }
```

Property Value

bool

NistEnabled

Controls whether or not to run the Nist process

```
bool NistEnabled { get; set; }
```

Property Value

bool

See Also

UpdateProperties()

Namespace EdaIntegrationContract.Import

Classes

ImportConstants

Constant values used during import processing

Interfaces

ICustodian

The party responsible for the source documents of a case

ICustodianManager

Manager of all interactions with the custodians of a case

IImportManager

Manager of all interactions for importing documents into a case

IImportProperties

Represents the import properties for a case.

IImportSet

An organization of sources imported into Explore or Turbo Import

IImportSetBuilder

A set of sources, with their respective custodians, that will be imported together.

IImportSetManager

Manager of all the Import Sets in a case

ISource

A file or folder identified for processing

ISourceInfo

Class containing info for the source grid in TurboImport

ISourceManager

Manager of all interactions for the document sources of a case

Enums

SourceType

The various types of sources

Interface ICustodian

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

The party responsible for the source documents of a case

```
public interface ICustodian : IEdaIntegrationUniqueEntity, IEdaIntegrationEntity
```

Inherited Members

IEdaIntegrationUniqueEntity.Id

Properties

LastUpdated

The timestamp of the last update in the database. This will be used to determine whether updates can be applied or if the user will need to refresh the data before updates can take place.

```
DateTime LastUpdated { get; }
```

Property Value

DateTime

Remarks

All date/time values are stored in UTC

Name

The name of the custodian.

```
string Name { get; set; }
```

Property Value

string

Methods

Equals(ICustodian)

Determines if two Custodians are the same (using id comparison)

```
bool Equals(ICustodian other)
```

Parameters

`other` ICustodian

The custodian to compare against

Returns

bool

True if the custodians are the same

Interface ICustodianManager

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

Manager of all interactions with the custodians of a case

```
public interface ICustodianManager
```

Methods

All()

Retrieves the list of Custodians for a case

```
IEnumerable<ICustodian> All()
```

Returns

IEnumerable<ICustodian>

A list of ICustodian

Examples

The following example demonstrates retrieving the a list of custodians for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        foreach (ICustodian custodian in edaCase.Custodians.All())
        {
            Console.WriteLine("Id: "   + custodian.Id);
            Console.WriteLine("Name: " + custodian.Name);
        }
    }
}
/*
This example produces the following results:

Id: 1
Name: Custodian 1
Id: 2
Name: Custodian 2
Id: 3
Name: Custodian 3
 */
```

ById(int)

Retrieves a Custodian by its unique identifier

```
ICustodian ById(int id)
```

Parameters

`id` int

The unique identifier of the custodian to find

Returns

ICustodian

The custodian with the supplied unique identifier

Examples

The following example demonstrates retrieving a custodian using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;
```

```
 class Sample
 {
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        ICustodian custodian = edaCase.Custodians.ById(3);
        Console.WriteLine("Id: {0},  Name: {1}", custodian.Id, custodian.Name);
    }
}
/*
This example produces the following results:

Id: 3,  Name: Custodian 3
 */
```

Exceptions

EdaApiException

Thrown if a custodian cannot be found with the specified Id

ByImportSet(int)

Retrieves the list of Custodians that had input in the supplied Import Set

```
IEnumerable<ICustodian> ByImportSet(int importSetId)
```

Parameters

`importSetId` int

The identifier of the import set to find

Returns

IEnumerable<ICustodian>

A list of ICustodians

Examples

The following example demonstrates retrieving a custodian using its name.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        foreach (ICustodian custodian in edaCase.Custodians.ByImportSet(123))
        {
            Console.WriteLine("Id: "   + custodian.Id);
            Console.WriteLine("Name: " + custodian.Name);
        }
    }
}
/*
This example produces the following results:

Id: 1
Name: Custodian 1
Id: 3
Name: Custodian 3
 */
```

Exceptions

EdaApiException

Thrown if a custodian cannot be found with the specified name

ByName(string)

Retrieves a Custodian by its name

```
ICustodian ByName(string custodianName)
```

Parameters

`custodianName` string

The name of the custodian to find

Returns

ICustodian

A list of ICustodians

Examples

The following example demonstrates retrieving a custodian using its name.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        ICustodian custodian = edaCase.Custodians.ByName("Custodian 2");
        Console.WriteLine("Id: {0},  Name: {1}", custodian.Id, custodian.Name);
    }
}
/*
This example produces the following results:

Id: 2,  Name: Custodian 2
 */
```

Exceptions

EdaApiException

Thrown if a custodian cannot be found with the specified name

Create(string)

Creates a new custodian for a case

```
ICustodian Create(string custodianName)
```

Parameters

`custodianName` string

The name of the new custodian

Returns

ICustodian

The custodian created

Examples

This example demonstrates how to create a new custodian for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        ICustodian custodian = edaCase.Custodians.Create("A new custodian");
        Console.WriteLine("Id: {0},  Name: {1}", custodian.Id, custodian.Name);
    }
}
/*
This example produces the following results:

Id: 4,  Name: A new custodian
 */
```

Exceptions

EdaApiException

Thrown if the custodian name is invalid

Delete(List<int>)

Delete one more custodians

> **WARNING**
>
> Deleting a custodian will also delete all the sources for each custodian, and by extension all the documents and metadata for those sources.

```
void Delete(List<int> custodianIdsToDelete)
```

Parameters

`custodianIdsToDelete` List<int>

The list of unique identifiers of the custodians to be deleted

Examples

This example demonstrates deleting a custodian.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Retrieve an existing custodian
        ICustodian custodian = edaCase.Custodians.ByName("Custodian 3");
        Console.WriteLine("({0}) {1}", custodian.Id, custodian.Name);

        // Delete the custodian
        List<int> custodiansToDelete = new List<int> { custodian.Id };
        edaCase.Custodians.Delete(custodiansToDelete);
        Console.WriteLine("Custodian deleted");

        // Attempting to find the custodian again will fail
        try
        {
          custodian = edaCase.Custodians.ByName("Custodian 3");
        }
        catch (EdaApiException ex)
        {
          Console.WriteLine("Exception: {0}", ex.Message);
        }
    }
}
/*
This example produces the following results:

Id: 3,  Name: Custodian 3
Custodian deleted
Exception: A custodian with the specified Name does not exist - Custodan 3
 */
```

RefreshNeeded(ref DateTime)

Checks the supplied timestamp against the one in the timestamps table for this repository returns true if the db timestamp is newer than the supplied timestamp and updates the supplied timestamp to the current db value

```
bool RefreshNeeded(ref DateTime previousTimestamp)
```

Parameters

`previousTimestamp` DateTime

A timestamp against which to compare the current last updated timestamp

Returns

bool

true if the db timestamp is newer than the supplied timestamp

Update(ICustodian)

Updates the system with any changed values for the custodian (also see Remarks below)

```
void Update(ICustodian custodian)
```

Parameters

`custodian` ICustodian

The custodian containing the updated data to save

Examples

This example demonstrates updating the name of an existing custodian.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Retrieve the existing custodian
        ICustodian custodian = edaCase.Custodians.ByName("Custodian 3");
        Console.WriteLine("({0}) {1}", custodian.Id, custodian.Name);

        // Change its name and save it back
        custodian.Name = "Third custodian";
        edaCase.Custodians.Update(custodian);
        Console.WriteLine("({0}) {1}", custodian.Id, custodian.Name);

        // Attempting to lookup the custodian using the old name
        // will fail now
        try
        {
            ICustodian prevCustodian = edaCase.Custodians.ByName("Custodian 3");
        }
        catch (EdaApiException ex)
        {
            Console.WriteLine("Exception: {0}", ex.Message);
        }
    }
}
/*
This example produces the following results:

Id: 3,  Name: Custodian 3
Id: 3,  Name: Third custodian
Exception: A custodian with the specified Name does not exist - Custodan 3
 */
```

Remarks

If any documents belonging to the custodian have already been indexed, and the custodian's name is changed, then the case will automatically be queued to re-index so that the index can be refreshed with the new custodian name.

Exceptions

EdaApiException

Thrown if the custodian name is invalid

ValidateCustodianName(string)

Validates that a custodian name passes all validation rules

```
void ValidateCustodianName(string custodianName)
```

Parameters

`custodianName` string

The name of the custodian to validate

Interface IImportManager

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

Manager of all interactions for importing documents into a case

```
public interface IImportManager
```

Properties

ImportSets

Provides access to the Turbo Import Import Sets for this case.

```
IImportSetManager ImportSets { get; }
```

Property Value

IImportSetManager

The *ImportSetManager* for accessing information relating to the processing Import Sets for this case.

Properties

Provides access to the Import properties for this case.

```
IImportProperties Properties { get; }
```

Property Value

IImportProperties

The *ImportProperties* for accessing analysis related properties for this case.

Methods

UpdateProperties()

Saves the import properties to the case.

```
void UpdateProperties()
```

Examples

The following example demonstrates updating the properties for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the MaxAnalysisAgents value
        edaCase.Imports.Properties.MaxAnalysisAgents = 4;

        // Save the changes to the case
        edaCase.Imports.UpdateProperties();
    }
}
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IImportProperties

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

Represents the import properties for a case.

```
public interface IImportProperties
```

Properties

AssignSuspectExtensions

Enable or disable assigning suspect extensions

```
bool AssignSuspectExtensions { get; set; }
```

Property Value

bool

ExpandCompoundDocuments

Enable or disable expanding compound documents

```
bool ExpandCompoundDocuments { get; set; }
```

Property Value

bool

ExtractCustomMetadataOffice

Extracts the custom metadata in Office documents.

```
bool ExtractCustomMetadataOffice { get; set; }
```

Property Value

bool

ExtractExifMetadata

Extracts the EXIF metadata from Image documents.

```
bool ExtractExifMetadata { get; set; }
```

Property Value

bool

IdentifyHiddenText

Enable or disable identify hidden text extraction

```
bool IdentifyHiddenText { get; set; }
```

Property Value

bool

LanguageRecognitionCommonLanguages

Enable or disable language identification of common languages only

```
bool LanguageRecognitionCommonLanguages { get; set; }
```

Property Value

bool

LanguageRecognitionEnabled

Enable or disable language identification

```
bool LanguageRecognitionEnabled { get; set; }
```

Property Value

bool

MaxAnalysisAgents

Gets and sets maximum number of analysis agents allowed to run on this case.

```
int MaxAnalysisAgents { get; set; }
```

Property Value

int

Examples

This example demonstrates how to set and retrieve the MaxAnalysisAgents.

```
using Law.EdaIntegration;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the MaxExportAgents value
        edaCase.Imports.Properties.MaxAnalysisAgents = 4;

        // Save the changes to the properties
        edaCase.Imports.UpdateProperties();

        // Get the value
        Console.WriteLine("MaxAnalysisAgents: {0}",
edaCase.Imports.Properties.MaxAnalysisAgents);
    }
}
/*
This example produces the following results:

MaxAnalysisAgents: 4
 */
```

Remarks

A value of 0 represents an unlimited number of agents.

Exceptions

EdaApiException

An exception is thrown if the max analysis agents value is not between 0 and 999

MaxPasswords

Maximum number passwords that can be added

```
int MaxPasswords { get; }
```

Property Value

int

Passwords

Provide passwords used in analysis and extraction operations.

```
IEnumerable<string> Passwords { get; set; }
```

Property Value

IEnumerable<string>

Examples

This example demonstrates how to set and retrieve the list of Passwords.

```
using Law.EdaIntegration;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create the list of passwords
        List<string> passwords = new List<string> { "abcdef", "123456", "qwerty" };

        // Set the Passwords list
        edaCase.Imports.Properties.Passwords = passwords;

        // Save the changes to the case
        edaCase.Imports.UpdateProperties();

        // Get the value
        foreach (string pwd in edaCase.Imports.Properties.Passwords)
        {
            Console.WriteLine("Password: {0}", pwd);
        }
    }
}
/*
This example produces the following results:

Password: abcdef
Password: 123456
Password: qwerty
 */
```

Remarks

Duplicate and blank passwords will be removed from the list when set. There is no limit to the number of passwords that can be defined but as the number of passwords increase, performance of the analysis and extraction process will decrease. Items in this list are case-sensitive.

See Also

UpdateProperties()

Interface IImportSet

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

An organization of sources imported into Explore or Turbo Import

```
public interface IImportSet : IEdaIntegrationUniqueEntity, IEdaIntegrationEntity
```

Properties

Created

The date and time that the ImportSet was created

```
DateTime? Created { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Id

The unique identifier for the ImportSet

```
int Id { get; }
```

Property Value

int

Label

A label applied to the custodian and sources being imported by the Import Set.

```
string Label { get; }
```

Property Value

string

Remarks

Import Set labels can be up to 100 characters and must be unique to the case. Any name longer than that will be truncated to 100 characters.

When an Import Set is created, if no label is specified, then a default value will be automatically generated using the following format: Import Set NNN, e.g. Import Set 001

Interface IImportSetBuilder

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

A set of sources, with their respective custodians, that will be imported together.

```
public interface IImportSetBuilder
```

Properties

ImportSetLabel

A label applied to the set of sources being imported.

```
string ImportSetLabel { get; set; }
```

Property Value

string

Remarks

Import Set labels can be up to 100 characters and must be unique to the case. Any name longer than that will be truncated to 100 characters.

When an Import Set is created, if no label is specified then a default value will be automatically generated.

IsCommitted

Identifies whether the Import Set has been submitted or not

```
bool IsCommitted { get; }
```

Property Value

bool

Methods

AddCustodian(int)

Adds an existing custodian to the import set

```
ICustodian AddCustodian(int custodianId)
```

Parameters

`custodianId` int

The identifier of the custodian

Returns

ICustodian

The ICustodian that was just added to the import set

AddCustodian(string)

Adds a custodian without sources to the import set. If a custodian with the provided name already exists then that custodian is returned.

```
ICustodian AddCustodian(string custodianName)
```

Parameters

`custodianName` string

The name of the custodian to add

Returns

ICustodian

The ICustodian that was just added to the import set

AddSources(List<string>, ICustodian, string)

Adds a list of files or folders to the Import Set.

```
IReadOnlyList<ISource> AddSources(List<string> filesAndFolders, ICustodian custodian, string
fileListName = "")
```

Parameters

`filesAndFolders` List<string>

A list of full-path file or folder names

`custodian` ICustodian

The custodian responsible for the data.

`fileListName` string

Optional name of a file list file, to consolidate all sources under that name

Returns

IReadOnlyList<ISource>

A list of the ISources that will be created or updated

AddSources(List<string>, bool, string)

Adds a list of files or folders to the Import Set.

```
IReadOnlyList<ISource> AddSources(List<string> filesAndFolders, bool
createMultipleCustodians, string fileListName = "")
```

Parameters

`filesAndFolders` List<string>

A list of full-path file or folder names

`createMultipleCustodians` bool

Identifies whether to create a single custodian to be responsible for the imported data or to create a custodian for each top-level item provided.

`fileListName` string

Optional name of a file list file, to consolidate all sources under that name

Returns

IReadOnlyList<ISource>

A list of the ISources that will be created or updated

Custodians()

The custodians that have data in the Import Set

```
IReadOnlyList<ICustodian> Custodians()
```

Returns

IReadOnlyList<ICustodian>

A read-only list of the ICustodians involved with this Import Set

RemoveCustodian(ICustodian)

Removes the custodian and all of its sources from the Import Set

```
void RemoveCustodian(ICustodian custodian)
```

Parameters

`custodian` ICustodian

The custodian to remove

RemoveSources(List<ISource>)

Removes sources from the Import Set. If the source being removed is the last source for new custodian then the custodian will also be removed.

```
void RemoveSources(List<ISource> uncommittedSources)
```

Parameters

`uncommittedSources` List<ISource>

The list of ISources to remove

Sources()

The sources to be included for the provided custodian when the Import Set is committed.

```
IReadOnlyList<ISource> Sources()
```

Returns

IReadOnlyList<ISource>

A read-only list of the ISources involved with this Import Set

Sources(ICustodian)

The sources to be included for the provided custodian when the Import Set is committed.

```
IReadOnlyList<ISource> Sources(ICustodian custodian)
```

Parameters

`custodian` ICustodian

The custodian of the sources

Returns

IReadOnlyList<ISource>

A read-only list of the ISources involved with this Import Set

Sources(int)

The sources to be included for the provided custodian when the Import Set is committed.

```
IReadOnlyList<ISource> Sources(int custodianId)
```

Parameters

`custodianId` int

The identifier of the custodian of the sources

Returns

IReadOnlyList<ISource>

A read-only list of the ISources involved with this Import Set

UpdateCustodianName(int, string)

Change the name of a custodian associated with the Import Set

```
ICustodian UpdateCustodianName(int custodianId, string newCustodianName)
```

Parameters

`custodianId` int

The unique identifier of the custodian to change

`newCustodianName` string

THe new name to use of for the custodian

Returns

ICustodian

The updated custodian with the new name

Exceptions

EdaApiException

Thrown if the custodian name is invalid or already in use

Interface IImportSetManager

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

Manager of all the Import Sets in a case

```
public interface IImportSetManager
```

Methods

All()

Retrieves the list of all the Import Sets in a case

```
IEnumerable<IImportSet> All()
```

Returns

IEnumerable<IImportSet>

A list of IImportSet

Examples

The following example demonstrates retrieving the a list of Import Sets for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        foreach (IImportSet importSet in edaCase.Imports.ImportSets.All())
        {
            Console.WriteLine("Label:   {0}", importSet.Label);
            Console.WriteLine("Created: {0}", importSet.Created);
            Console.WriteLine();
        }
    }
}
/*
This example produces the following results:

Label:   2014-06-19-001
Created: 06/19/2004 9:10:22 AM

Label:   2014-06-21-001
Created: 06/21/2004 10:13:12 AM

Label:   2014-06-21-002
Created: 06/21/2004 6:23:45 PM

Label:   Local Data
Created: 06/21/2004 6:24:19 PM

 */
```

ById(int)

Retrieves a specific Import Set in a case using its unique identifier

```
IImportSet ById(int importSetId)
```

Parameters

`importSetId` int

The unique identifier of the import set

Returns

IImportSet

IImportSet

Examples

The following example demonstrates retrieving an import set using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        IImportSet importSet = edaCase.Imports.ImportSets.ById(3);
        Console.WriteLine("Id: {0},  Label: {1}", importSet.Id, importSet.Label);
    }
}
/*
This example produces the following results:

Id: 3,  Label: 2014-06-21-002
 */
```

ByLabel(string)

Retrieves a specific import set in a case using its label

```
IImportSet ByLabel(string importSetLabel)
```

Parameters

`importSetLabel` string

The label of the import set

Returns

IImportSet

IImportSet

Examples

The following example demonstrates retrieving an import set using its label.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        IImportSet importSet = edaCase.Imports.ImportSets.ByLabel("Local Data");
        Console.WriteLine("Id: {0},  Label: {1}", importSet.Id, importSet.Label);
    }
}
/*
This example produces the following results:

Id: 4,  Label: Local Data
 */
```

GetNewBuilder()

Retrieves a new ImportSetBuilder which can be used to describe the sources that need to be imported.

```
IImportSetBuilder GetNewBuilder()
```

Returns

IImportSetBuilder

IImportSetBuilder

Examples

The following example demonstrates how to retrieve an ImportSetBuilder and use it to import data.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        var inputFiles = new List<string> ()
        {
              @"C:\Temp\SomeFolder\Data1",
              @"C:\Temp\SomeFolder\Data2",
              @"C:\Temp\AnotherFolder\Data",
        };

        IImportSetBuilder builder = edaCase.Imports.ImportSets.GetNewBuilder();
        builder.AddSources(inputFiles, true);
        edaCase.Imports.ImportSets.StartImport(builder);
    }
}
```

StartImport(IImportSetBuilder)

Starts the processing to import and analyze any new document sources added to the case

```
void StartImport(IImportSetBuilder builder)
```

Parameters

`builder` IImportSetBuilder

The IImportSetBuilder which describes the data to use for the import processing

Examples

The following example demonstrates how to start the import processing for a case

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        var inputFiles = new List<string> ()
        {
              @"C:\Temp\SomeFolder\Data1",
              @"C:\Temp\SomeFolder\Data2",
              @"C:\Temp\AnotherFolder\Data",
        };

        IImportSetBuilder builder = edaCase.Imports.ImportSets.GetNewBuilder();
        builder.AddSources(inputFiles, true);
        edaCase.Imports.ImportSets.StartImport(builder);
    }
}
```

Interface ISource

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

A file or folder identified for processing

```
public interface ISource : IEdaIntegrationEntity
```

Properties

AnalysisState

Analysis state

```
AnalysisStates AnalysisState { get; }
```

Property Value

AnalysisStates

CustodianId

The unique identifier of the Custodian that is responsible for this source.

```
int CustodianId { get; }
```

Property Value

int

Description

An auto-generated description describing the inputs for the source.

```
string Description { get; }
```

Property Value

string

Remarks

The description for a source is automatically generated when the source is created. The input files/folders are used to build the descrition as follows:

Source Type	Description Value	Example
Folder	The full path to the folder	\\Server\Samples\FolderA
File	The full path to the folder containing the files followed by the number of files	\\Server\Samples\Documents (10 files)

Id

The unique identifier for the source

```
int Id { get; }
```

Property Value

int

ImportSetId

The unique identifier of the Import Set within which this source was imported.

```
int ImportSetId { get; }
```

Property Value

int

Input

Input Field

```
string Input { get; }
```

Property Value

string

InventoryState

Inventory state

```
InventoryStates InventoryState { get; }
```

Property Value

InventoryStates

Name

The name (or label) of the source.

```
string Name { get; set; }
```

Property Value

string

Remarks

A source name can be up to 255 characters.

If a name is not supplied when a source is created, a default value will be generated. The default value will be the last directory in the full path of the folder or files contained in the source.

Source Type	Path	Default Name
Folder	\\Server\Share\Folder1\Folder2\Folder3	Folder3
File	\\Server\Share\FolderA\FolderB\document.docx	FolderB

SourceState

Source state

```
SourceStates SourceState { get; }
```

Property Value

SourceStates

Type

Identifies whether the source is a folder or a set of files

```
SourceType Type { get; }
```

Property Value

SourceType

Interface ISourceInfo

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

Class containing info for the source grid in TurboImport

```
public interface ISourceInfo : IEdaIntegrationUniqueEntity, IEdaIntegrationEntity
```

Properties

AnalysisState

Analysis state

```
string AnalysisState { get; }
```

Property Value

string

CustodianId

The unique identifier of the Custodian that is responsible for this source.

```
int CustodianId { get; }
```

Property Value

int

CustodianName

Custodian name

```
string CustodianName { get; }
```

Property Value

string

Description

An auto-generated description describing the inputs for the source.

```
string Description { get; }
```

Property Value

string

Id

The unique identifier for the source

```
int Id { get; }
```

Property Value

int

ImportSetId

The unique identifier of the Import Set within which this source was imported.

```
int ImportSetId { get; }
```

Property Value

int

ImportSetName

Import Set name

```
string ImportSetName { get; }
```

Property Value

string

Input

Input Field

```
string Input { get; }
```

Property Value

string

InventoryState

Inventory state

```
string InventoryState { get; }
```

Property Value

string

Name

The name (or label) of the source.

```
string Name { get; set; }
```

Property Value

string

SourceState

Source state

```
string SourceState { get; }
```

Property Value

string

SourceStatistics

Source Statistics

```
ISourceStatistics SourceStatistics { get; set; }
```

Property Value

ISourceStatistics

Type

Identifies whether the source is a folder or a set of files

```
string Type { get; }
```

Property Value

string

Interface ISourceManager

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

Manager of all interactions for the document sources of a case

```
public interface ISourceManager
```

Methods

All()

Retrieves the list of Sources for a case

```
IEnumerable<ISource> All()
```

Returns

IEnumerable<ISource>

A list of ISources

Examples

The following example demonstrates retrieving the a list of sources for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        foreach (ISource s in edaCase.Sources.All())
        {
            Console.WriteLine("Id: " + s.Id);
            Console.WriteLine("Name: " + s.Name);
            Console.WriteLine("Type: " + s.Type);
            Console.WriteLine("CustodianId: " + s.CustodianId);
            Console.WriteLine("Description: " + s.Description);
            Console.WriteLine("");
        }
    }
}
/*
This example produces the following results:

  Id: 15
  Name: Documents
  Type: Folder
  CustodianId: 4
  Description: \\NetworkDrive\Folder1\Folder2\Documents

  Id: 16
  Name: Folder 1
  Type: Files
  CustodianId: 4
  Description: \\NetworkDrive\Folder1 (3 files)

 */
```

AllSourceInfos()

Returns a list of sourceinfo objects, can be used for informational display of sources

```
IEnumerable<ISourceInfo> AllSourceInfos()
```

Returns

IEnumerable<ISourceInfo>

A list of ISourceInfo containing information about the sources in the case

ByCustodian(int)

Retrieves the list of Sources for a particular custodian

```
IEnumerable<ISource> ByCustodian(int custodianId)
```

Parameters

`custodianId` int

The unique identifier of the custodian

Returns

IEnumerable<ISource>

A list of ISources associated with a Custodian

Examples

The following example demonstrates retrieving the a list of sources for a custodian.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        int custodianId = 4;
        foreach (ISource s in edaCase.Sources.ByCustodian(custodianId))
        {
            Console.WriteLine("Id: " + s.Id);
            Console.WriteLine("Name: " + s.Name);
            Console.WriteLine("Type: " + s.Type);
            Console.WriteLine("CustodianId: " + s.CustodianId);
            Console.WriteLine("Description: " + s.Description);
            Console.WriteLine("");
        }
    }
}
/*
This example produces the following results:
```

```
Id: 15
Name: Documents
Type: Folder
CustodianId: 4
Description: \\NetworkDrive\Folder1\Folder2\Documents

Id: 16
Name: Folder 1
Type: Files
CustodianId: 4
Description: \\NetworkDrive\Folder1 (3 files)

 */
```

ById(int)

Retrieves a Source by its unique identifier

```
ISource ById(int id)
```

Parameters

`id` int

The unique identifier of the source to find

Returns

ISource

A list of ISources

Examples

The following example demonstrates retrieving Source using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        ISource source = edaCase.Sources.ById(15);
        Console.WriteLine("Id: " + source.Id);
        Console.WriteLine("Name: " + source.Name);
        Console.WriteLine("Type: " + source.Type);
        Console.WriteLine("CustodianId: " + source.CustodianId);
        Console.WriteLine("Description: " + source.Description);
    }
}
/*
This example produces the following results:

Id: 15
Name: Documents
Type: Folder
CustodianId: 4
Description: \\NetworkDrive\Folder1\Folder2\Documents
 */
```

Exceptions

EdaApiException

Thrown if a source cannot be found with the specified Id

ByImportSet(int, int)

Retrieves the list of Sources for an ImportSet

```
IEnumerable<ISource> ByImportSet(int importSetId, int custodianId = 0)
```

Parameters

`importSetId` int

The unique identifier of the Import Set

`custodianId` int

Returns

IEnumerable<ISource>

A list of ISources

Examples

The following example demonstrates retrieving the sources for an Import Set

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        foreach(ISource source in edaCase.Sources.ByImportSet(3))
        {
            Console.WriteLine("Id: " + source.Id);
            Console.WriteLine("Name: " + source.Name);
            Console.WriteLine("Type: " + source.Type);
            Console.WriteLine("CustodianId: " + source.CustodianId);
            Console.WriteLine("Description: " + source.Description);
        }
    }
}
/*
This example produces the following results:

Id: 15
Name: Petes Data
Type: Folder
CustodianId: 4

Id: 16
Name: Bobs Data
Type: Folder
CustodianId: 5
 */
```

ByName(string)

Retrieves a Source by its name

```
ISource ByName(string name)
```

Parameters

name string

The name of the source to find

Returns

ISource

A list of ISources

Examples

The following example demonstrates retrieving Source using its name.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        ISource source = edaCase.Sources.ByName("Documents");
        Console.WriteLine("Id: " + source.Id);
        Console.WriteLine("Name: " + source.Name);
        Console.WriteLine("Type: " + source.Type);
        Console.WriteLine("CustodianId: " + source.CustodianId);
        Console.WriteLine("Description: " + source.Description);
    }
}
/*
This example produces the following results:

Id: 15
```

```
Name: Documents
Type: Folder
CustodianId: 4
Description: \\NetworkDrive\Folder1\Folder2\Documents
 */
```

Exceptions

EdaApiException

Thrown if a source cannot be found with the specified name

Delete(IEnumerable<int>)

Deletes one or more sources

> **WARNING**
> When deleting a source, the documents and metadata associated with that source will also be deleted.

```
void Delete(IEnumerable<int> sourceIdsToDelete)
```

Parameters

sourceIdsToDelete IEnumerable<int>

The list of unique identifiers of the sources to be deleted

Examples

The following example demonstrates deleting a list of sources.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Import;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        List<int> sourceIdsToDelete = new List<int>
        {
            15,
            21,
        };
        edaCase.Sources.Delete(sourceIdsToDelete);
    }
}
```

Class ImportConstants

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

Constant values used during import processing

```
public static class ImportConstants
```

Inheritance

object ← ImportConstants

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

MaxCustodianNameLength

The maximum length of a custodian's name

```
public const int MaxCustodianNameLength = 50
```

Field Value

int

Enum SourceType

Namespace: EdaIntegrationContract.Import

Assembly: EdaIntegration.Contract.dll

The various types of sources

```
public enum SourceType
```

Fields

`Files = 1`

One or more full path files to load

`Folder = 0`

A folder to crawl and discover files to load

Namespace EdaIntegrationContract.Indexing

Interfaces

IIndexManager

Manager of all interactions for the indexing of a case

IIndexProperties

The document properties that will be indexed for a case.

Enums

HyphenTreatmentType

Possible ways that hyphens can be treated during indexing.

Enum HyphenTreatmentType

Namespace: EdaIntegrationContract.Indexing

Assembly: EdaIntegration.Contract.dll

Possible ways that hyphens can be treated during indexing.

```
public enum HyphenTreatmentType
```

Fields

`AllThree = 3`

For example, first-class is indexed as first-class, firstclass, first, and class.

`HypensAsSpaces = 0`

For example, first-class is indexed as first and class. (default)

`HyphensAsSearchable = 1`

For example, first-class is indexed as first-class.

`HyphensIgnored = 2`

For example, first-class is indexed as firstclass.

Interface IIndexManager

Namespace: EdaIntegrationContract.Indexing

Assembly: EdaIntegration.Contract.dll

Manager of all interactions for the indexing of a case

```
public interface IIndexManager
```

Properties

Properties

Provides access to the Index properties for this case.

```
IIndexProperties Properties { get; }
```

Property Value

IIndexProperties

The *IndexProperties* for accessing indexing-related properties for this case.

Methods

UpdateProperties()

Saves the indexing properties for the case.

```
void UpdateProperties()
```

Examples

The following example demonstrates updating the indexing properties for a case.

```csharp
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Indexing;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        edaCase.Index.Properties.IndexingEnabled = true;
        edaCase.Index.Properties.MaxIndexingWorkers = 4;

        // Save the changes to the case
        edaCase.Index.UpdateProperties();
    }
}
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IIndexProperties

Namespace: EdaIntegrationContract.Indexing

Assembly: EdaIntegration.Contract.dll

The document properties that will be indexed for a case.

```
public interface IIndexProperties
```

Properties

AccentSensitive

Indicates whether to treat accents as significant when comparing letters.

```
bool AccentSensitive { get; set; }
```

Property Value

bool

Remarks

For most situations this is not recommended because this option increases the chance of missing the retrieval of a document if an accent was omitted in one letter.

AutoRecognizeDates

Indicates that the indexer should automatically scan for anything that looks like a date, e-mail address or credit card number.

```
bool AutoRecognizeDates { get; set; }
```

Property Value

bool

CaseSensitive

Indicates whether the indexer will take capitalization into account in indexing words.

```
bool CaseSensitive { get; set; }
```

Property Value

bool

Remarks

In a case-sensitive index, "CREDIT", "Credit", and "credit" would be three different words. This option can be useful, for example, when you are searching for a term, such as a capitalized name that can be confused with a common non-capitalized word.

HyphenTreatment

Indicates how hyphenated words should be treated during indexing.

```
HyphenTreatmentType HyphenTreatment { get; set; }
```

Property Value

HyphenTreatmentType

IndexingEnabled

Indicates whether to run indexing automatically when documents are added to the case.

```
bool IndexingEnabled { get; set; }
```

Property Value

bool

MaxIndexingWorkers

The maximum number of processes that will be used during the indexing process, up to a maximum of 32.

Valid vales are 0-8, 12, 16, 20, 24, 28, 32

```
int MaxIndexingWorkers { get; set; }
```

Property Value

int

Remarks

A value of 0 means that the system will use 1 worker per available processor, with a limit of 8 workers. If a machine has more than 8 processor then only a maximum of 8 workers will be used.

MaxSearchWorkers

The maximum number of processes that will be used during search processing, up to a maximum of 16.

Valid vales are 0-8, 12, 16

```
int MaxSearchWorkers { get; set; }
```

Property Value

int

Remarks

A value of 0 means that the system will use 1 worker per available processor, with a limit of 8 workers. If a machine has more than 8 processor then only a maximum of 8 workers will be used.

ReindexingRequired

Indicates that an indexing property was changed that requires reindexing all documents.

```
bool ReindexingRequired { get; }
```

Property Value

bool

UseLocalTempFolder

Indicates whether indexing should use a local temporary folder for temporary files created during the indexing process.

```
bool UseLocalTempFolder { get; set; }
```

Property Value

bool

Remarks

During indexing, the dtSearch Engine may need to create temporary files to store word lists that are too large to fit into memory. By default, these files will be placed in the index folder. Use this setting to instruct the indexer to use the local machine's temporary folder for these files. The indexer will automatically delete the temporary files when the index update completes.

WorkBreakCJK

Indicates whether the indexer should insert work breaks between Chinese, Japanese and Korean characters.

```
bool WorkBreakCJK { get; set; }
```

Property Value

bool

Remarks

Use this option if you plan to search Chinese, Japanese, or Korean documents that do not contain word breaks. Some Chinese, Japanese, and Korean text does not include word breaks. Instead, the text appears as lines of characters with no spaces between the words. Because there are no spaces separating the words on each line, the indexer sees each line of text as a single long word. To make this type of text searchable, enable automatic insertion of word breaks around Chinese, Japanese, and Korean characters so each character will be treated as single word.

Namespace EdaIntegrationContract.NearDuplicate

Interfaces

IExcludedContentItem

Represents an excluded content item.

IExcludedContentManager

Represents a class for managing excluded content items for a case.

INearDupeMetadata

Document metadata regarding Near Duplicate analysis

INearDuplicateManager

Represents a class for managing the near-duplicate identification process.

INearDuplicateProperties

Represents the near-duplicate identification properties for a case.

Interface IExcludedContentItem

Namespace: EdaIntegrationContract.NearDuplicate

Assembly: EdaIntegration.Contract.dll

Represents an excluded content item.

```
public interface IExcludedContentItem
```

Remarks

Also referred to as a footer exclusion item used to exclude repeated content during the near duplication process.

Properties

DateModified

The time stamp of the last update in the database.

```
DateTime? DateModified { get; }
```

Property Value

DateTime?

Remarks

All date/time values are stored in UTC

Id

The unique identifier of the excluded item.

```
int Id { get; }
```

Property Value

int

Phrase

The text or phrase of the excluded content item.

```
string Phrase { get; set; }
```

Property Value

string

Remarks

This is commonly repeated text such as legal disclaimers found at the bottom of e-mails which can cause the near duplication process to make two or more documents to appear more similar then they actually are.

See Also

IExcludedContentManager

Interface IExcludedContentManager

Namespace: EdaIntegrationContract.NearDuplicate

Assembly: EdaIntegration.Contract.dll

Represents a class for managing excluded content items for a case.

```
public interface IExcludedContentManager
```

Remarks

Excluded content items can be used to exclude repeated content from the near-duplication processing. These items are also referred to as Footer Exclusions.

Methods

All()

Retrieves the list of all IExcludedContentItems for a case.

```
IEnumerable<IExcludedContentItem> All()
```

Returns

IEnumerable<IExcludedContentItem>

A list of IExcludedContentItems.

Examples

The following example demonstrates retrieving a list of excluded content items for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.NearDuplicate;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        foreach (IExcludedContentItem excludedItem in
edaCase.NearDuplicates.Properties.ExcludedContent.All())
        {
            Console.WriteLine("Id: "    + excludedItem.Id);
            Console.WriteLine("Phrase: " + excludedItem.Phrase);
            Console.WriteLine("DateModified: " + excludedItem.DateModified);
            Console.WriteLine();
        }
    }
}
/*
This example produces the following results:

Id: 1
Phrase: Some text A
DateModified: 8/26/2014 9:40:35 AM

Id: 2
Phrase: Some text B
DateModified: 8/26/2014 9:40:35 AM
 */
```

ById(int)

Retrieves an excluded content item by its unique identifier.

```
IExcludedContentItem ById(int id)
```

Parameters

id int

The unique identifier of the excluded content item to find.

Returns

IExcludedContentItem

The excluded content item with the supplied unique identifier.

Examples

The following example demonstrates retrieving an excluded content item using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.NearDuplicate;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create an excluded content item
        string phrase = "This e-mail may contain information that is privileged
or confidential.";
        IExcludedContentItem item =
edaCase.NearDuplicates.Properties.ExcludedContent.Create(phrase);

        // Retrieve the excluded content item and display it
        IExcludedContentItem itemById =
edaCase.NearDuplicates.Properties.ExcludedContent.ById(item.Id);
        Console.WriteLine("Id: {0}", itemById.Id);
        Console.WriteLine("Phrase: {0}", itemById.Phrase);
    }
}
/*
This example produces the following results:

Id: 1
Phrase: This e-mail may contain information that is privileged or confidential.
 */
```

Exceptions

EdaApiException

Thrown if an excluded content item cannot be found with the specified Id.

Create(string)

Creates a new excluded content item for a case.

```
IExcludedContentItem Create(string phrase)
```

Parameters

phrase string

The text or phrase of the excluded content item.

Returns

IExcludedContentItem

The excluded content item

Examples

The following example demonstrates how to create an excluded content item for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.NearDuplicate;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create an excluded content item
        string phrase = "This e-mail may contain information that is privileged 
or confidential.";
        IExcludedContentItem item = 
edaCase.NearDuplicates.Properties.ExcludedContent.Create(phrase);
        Console.WriteLine("Id: {0}", item.Id);
        Console.WriteLine("Phrase: {0}", item.Phrase);
    }
}
/*
This example produces the following results:
```

```
Id: 1
Phrase: This e-mail may contain information that is privileged or confidential.
 */
```

Remarks

If a duplicate phrase is added, the existing item is returned without creating a new record.

Delete(IEnumerable<int>)

Delete one more excluded content items.

```
void Delete(IEnumerable<int> idsToDelete)
```

Parameters

idsToDelete IEnumerable<int>

The list of unique identifiers of the excluded content items to be deleted.

Examples

This example demonstrates deleting an excluded content item.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.NearDuplicate;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

         // Create an excluded content item
         string phrase = "This e-mail may contain information that is privileged
or confidential.";
         IExcludedContentItem item =
edaCase.NearDuplicates.Properties.ExcludedContent.Create(phrase);
         Console.WriteLine("Id: {0}", item.Id);
```

```
          Console.WriteLine("Phrase: {0}",item.Phrase);

          // Delete the excluded content item
          List<int> itemsToDelete = new List<int> { item.Id };
          edaCase.NearDuplicates.Properties.ExcludedContent.Delete(itemsToDelete);
          Console.WriteLine("Excluded content item deleted");

          // Attempting to find the excluded content item again will fail
          try
          {
            item = edaCase.NearDuplicates.Properties.ExcludedContent.ById(item.Id);
          }
          catch (EdaApiException ex)
          {
            Console.WriteLine("Exception: {0}", ex.Message);
          }
     }
 }
 /*
 This example produces the following results:

 Id: 1
 Phrase: This e-mail may contain information that is privileged or confidential.
 Excluded content item deleted
 Exception: An excluded content item with the specified Id does not exist - 1
  */
```

Update(IExcludedContentItem)

Updates the system with any changed values for the excluded content item.

```
void Update(IExcludedContentItem excludedContentItem)
```

Parameters

`excludedContentItem` IExcludedContentItem

The excluded content item containing the updated data to save.

Examples

This example demonstrates changing an excluded content item.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.NearDuplicate;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create a new ExcludedContentItem
        string phrase = "This e-mail may contain information that is privileged
or confidential.";
        IExcludedContentItem item =
edaCase.NearDuplicates.Properties.ExcludedContent.Create(phrase);
        Console.WriteLine("Before");
        Console.WriteLine("({0}): {1}", item.Id, item.Phrase);

        // Change the phrase and save it back
        item.Phrase = "This e-mail contains information that is privileged
or confidential.";
        edaCase.NearDuplicates.Properties.ExcludedContent.Update(item);
        Console.WriteLine("After");
        Console.WriteLine("({0}): {1}", item.Id, item.Phrase);
    }
}
/*
This example produces the following results:

Before
(2): This e-mail may contain information that is privileged or confidential.
After
(2): This e-mail contains information that is privileged or confidential.
 */
```

Exceptions

EdaApiException

Thrown if an excluded content item cannot be found with the a matching Id.

Interface INearDupeMetadata

Namespace: EdaIntegrationContract.NearDuplicate

Assembly: EdaIntegration.Contract.dll

Document metadata regarding Near Duplicate analysis

```
public interface INearDupeMetadata
```

Properties

IsMaster

Indicates if this document is the "master" document in the near-dupe family.

```
bool IsMaster { get; }
```

Property Value

bool

Master

The Id of the "master" document in the near-dupe family. All document comparisons in the family are made to this document.

```
int Master { get; }
```

Property Value

int

Similarity

A numeric value representing how similar this document is to the "master" document in the near-dupe family.

```
int Similarity { get; }
```

Property Value

int

Interface INearDuplicateManager

Namespace: EdaIntegrationContract.NearDuplicate

Assembly: EdaIntegration.Contract.dll

Represents a class for managing the near-duplicate identification process.

```
public interface INearDuplicateManager
```

Properties

Properties

Provides access to the near-duplicate identification properties for this case.

```
INearDuplicateProperties Properties { get; }
```

Property Value

INearDuplicateProperties

Returns the *NearDuplicateProperties* for accessing near-duplicate identification related properties for this case.

Methods

UpdateProperties()

Saves the INearDuplicateProperties to the case.

```
void UpdateProperties()
```

Examples

The following example demonstrates updating the Near Duplicate properties for a case.

```
using Law.EdaIntegration;
using Law.EdaIntegration.Case;
using Law.EdaIntegration.NearDuplicate;

class Sample
{
    static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.ByName("Case 1");

        // Set the NearDupeComparisonThreshold value
        edaCase.NearDuplicates.Properties.NearDupeComparisonThreshold = 85;

        // Save the changes to the case
        edaCase.NearDuplicates.UpdateProperties();
    }
}
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface INearDuplicateProperties

Namespace: EdaIntegrationContract.NearDuplicate

Assembly: EdaIntegration.Contract.dll

Represents the near-duplicate identification properties for a case.

```
public interface INearDuplicateProperties
```

Properties

ExcludedContent

Represents the IExcludedContentManager class for managing excluded content items.

```
IExcludedContentManager ExcludedContent { get; set; }
```

Property Value

IExcludedContentManager

NearDupeComparisonThreshold

Gets or sets the Near-Duplicate Comparison Threshold for the case.

```
int NearDupeComparisonThreshold { get; set; }
```

Property Value

int

Examples

This example demonstrates how to set and retrieve the NearDupeComparisonThreshold.

```
using Law.EdaIntegration;

class Sample
{
    static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the NearDupeComparisonThreshold value
        edaCase.NearDuplicates.Properties.NearDupeComparisonThreshold = 85;

        // Save the changes to the case
        edaCase.NearDuplicates.UpdateProperties();

        // Get the value
        Console.WriteLine("NearDupeComparisonThreshold: {0}",
edaCase.NearDuplicates.Properties.NearDupeComparisonThreshold);
    }
}
/*
This example produces the following results:

NearDupeComparisonThreshold: 85
 */
```

Remarks

Determines the minimum percentage of similarity to be considered when comparing the similarity between case files during near-duplicate analysis. The valid range for this setting is from 60 to 99.

Exceptions

EdaApiException

Thrown if the value is not between 60 and 99.

NearDupeEnabled

Determines whether near-duplicate analysis is enabled for the case.

```
bool NearDupeEnabled { get; set; }
```

Property Value

bool

Examples

This example demonstrates how to set and retrieve the NearDupeEnabled property.

```
using Law.EdaIntegration;

class Sample
{
    static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the NearDupeEnabled value
        edaCase.NearDuplicates.Properties.NearDupeEnabled = true;

        // Save the changes to the case
        edaCase.NearDuplicates.UpdateProperties();

        // Get the value
        Console.WriteLine("NearDupeEnabled: {0}",
edaCase.NearDuplicates.Properties.NearDupeEnabled);
    }
}
/*
This example produces the following results:

NearDupeEnabled: True
 */
```

Remarks

When near-duplicate analysis is enabled for a case, the near-duplicate processing will automatically start if the system detects that near-duplicate analysis has never run on the case or if there are changes to the case's current data since the last time the analysis ran. Examples of changes that could cause near-duplicate analysis to be run include the import of additional documents, deletion of case documents, and changes to the Comparison Threshold or Footer Exclusions settings.

See Also

UpdateProperties()

Namespace EdaIntegrationContract.OCR

Classes

InvalidOcrEngineException

Exception generated from the Integration Library when an invalid OCR engine is requested.

Interfaces

IOcrEngine

OCR engine supported by the Integration Library

IOcrManager

Represents a class for managing OCR.

IOcrProperties

Represents the OCR properties for a case.

Enums

OcrEngineType

The OCR engines recognized by the Integration Library

Interface IOcrEngine

Namespace: EdaIntegrationContract.OCR

Assembly: EdaIntegration.Contract.dll

OCR engine supported by the Integration Library

```
public interface IOcrEngine
```

Properties

EngineType

The type of OCR engine

```
OcrEngineType EngineType { get; }
```

Property Value

OcrEngineType

Installed

Whether the selected OCR engine is installed on the machine.

```
bool Installed { get; }
```

Property Value

bool

IsActiveEngine

Indicates whether the OCR engine is currently active

```
bool IsActiveEngine { get; }
```

Property Value

bool

Interface IOcrManager

Namespace: EdaIntegrationContract.OCR

Assembly: EdaIntegration.Contract.dll

Represents a class for managing OCR.

```
public interface IOcrManager
```

Properties

ActiveOcrEngine

Retrieve the OCR engine that is currently active

```
IOcrEngine ActiveOcrEngine { get; }
```

Property Value

IOcrEngine

Properties

Provides access to the OCR properties for this case.

```
IOcrProperties Properties { get; }
```

Property Value

IOcrProperties

The *OcrProperties* for accessing OCR related properties for this case.

Methods

GetOcrEngine<T>()

Get an OCR engine

```
T GetOcrEngine<T>() where T : class, IOcrEngine
```

Returns

T

The desired OCR engine

Type Parameters

T

The type of OCR engine to retrieve

Examples

The following example demonstrates updating the properties for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.OCR;
using EdaIntegrationContract.OCR.Expervision;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set OCR case properties
        IOcrProperties ocrProps = theCase.OCR.Properties;
        ocrProps.MaxOcrAgents = 4;
        ocrProps.QueueForOcrDuringAnalysis = true;
        ocrProps.AnalysisOcrFileTypes = theCase.OCR.GetValidOcrFileTypes();
        theCase.OCR.UpdateProperties();

        // Specify which OCR engine to use
        IExpervisionOcrEngine ocrEng = theCase.OCR.GetOcrEngine<IExpervisionOcrEngine>();
        ocrEng.Language = ExpervisionLanguage.English;
```

```
            ocrEng.AutoDeskew = true;
            ocrEng.AutoRotate = ExpervisionAutoRotateSetting.AlwaysOn;
            theCase.OCR.SetActiveOcrEngine(ocrEng);
        }
    }
```

GetValidOcrFileTypes()

Get a list of the file types that support OCR.

```
List<string> GetValidOcrFileTypes()
```

Returns

List<string>

A list of file type Id's.

Examples

The following example demonstrates retrieving a list of valid file types for OCR.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.OCR;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        //Get the list of valid OCR file types
        List<string> ocrFileTypeIds = edaCase.OCR.GetValidOcrFileTypes();

        // Set the AnalysisOcrFileTypes list
        edaCase.OCR.Properties.AnalysisOcrFileTypes = ocrFileTypeIds;

        // Save the changes to the case
        edaCase.OCR.UpdateProperties();
    }
}
```

SetActiveOcrEngine(IOcrEngine)

Designate the supplied OCR engine as the one to use

```
void SetActiveOcrEngine(IOcrEngine ocrEngine)
```

Parameters

`ocrEngine` IOcrEngine

The OCR engine to use

Examples

The following example demonstrates updating the properties for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.OCR;
using EdaIntegrationContract.OCR.Expervision;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set OCR case properties
        IOcrProperties ocrProps = theCase.OCR.Properties;
        ocrProps.MaxOcrAgents = 4;
        ocrProps.QueueForOcrDuringAnalysis = true;
        ocrProps.AnalysisOcrFileTypes = theCase.OCR.GetValidOcrFileTypes();
        theCase.OCR.UpdateProperties();

        // Specify which OCR engine to use
        IExpervisionOcrEngine ocrEng = theCase.OCR.GetOcrEngine<IExpervisionOcrEngine>();
        ocrEng.Language = ExpervisionLanguage.English;
        ocrEng.AutoDeskew = true;
        ocrEng.AutoRotate = ExpervisionAutoRotateSetting.AlwaysOn;
        theCase.OCR.SetActiveOcrEngine(ocrEng);
    }
}
```

UpdateProperties()

Saves the OCR properties to the case.

```
void UpdateProperties()
```

Examples

The following example demonstrates updating the properties for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.OCR;

class Sample
{
    public static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the MaxOcrAgents value
        edaCase.OCR.Properties.MaxOcrAgents = 4;

        // Save the changes to the case
        edaCase.OCR.UpdateProperties();
    }
}
```

Remarks

Modifications to the properties are not saved to the case until UpdateProperties() is called.

Interface IOcrProperties

Namespace: EdaIntegrationContract.OCR

Assembly: EdaIntegration.Contract.dll

Represents the OCR properties for a case.

```
public interface IOcrProperties
```

Properties

AnalysisOcrFileTypes

Gets or sets a list of file types to get queued for OCR by the analyzer when the QueueForOcrDuring Analysis attribute is active.

```
List<string> AnalysisOcrFileTypes { get; set; }
```

Property Value

List<string>

Examples

The following example demonstrates setting a list of valid file types for OCR.

```
using System.Collections.Generic;
using Law.EdaIntegration;
using Law.EdaIntegration.Case;
using Law.EdaIntegration.Ocr;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        //Get the list of valid OCR file types
        List<string> ocrFileTypeIds =
Law.EdaIntegration.Ocr.OcrManager.GetValidOcrFileTypes();

        // Set the AnalysisOcrFileTypes list
        edaCase.OCR.Properties.AnalysisOcrFileTypes = ocrFileTypeIds;

        // Save the changes to the case
        edaCase.OCR.UpdateProperties();
    }
}
```

Remarks

List defaults to include all of the most common file types. A list of supported file type Id's can be found by using GetValidOcrFileTypes(). Duplicate file type Id's will be removed from the list automatically when set.

LimitOcrTextSize

Gets or sets whether or not to exclude PDF documents from OCR that have extracted text larger than a given size, when the QueueForOcrDuringAnalysis attribute is active.

```
bool LimitOcrTextSize { get; set; }
```

Property Value

bool

Examples

The following example demonstrates setting this property.

```
using Law.EdaIntegration;
using Law.EdaIntegration.Case;
using Law.EdaIntegration.Ocr;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");
        // Set LimitOcrTextSize and MaxOcrableTextSize
        edaCase.OCR.Properties.LimitOcrTextSize = true;
        edaCase.OCR.Properties.MaxOcrableTextSize = 200

        // Save the changes to the case
        edaCase.OCR.UpdateProperties();
    }
}
```

Remarks

Works in combination with the MaxOcrableTextSize property.

MaxOcrAgents

Gets and sets maximum number of OCR agents allowed to run on this case.

```
int MaxOcrAgents { get; set; }
```

Property Value

int

Examples

This example demonstrates how to set and retrieve the MaxOcrAgents.

```
using Law.EdaIntegration;
using Law.EdaIntegration.Case;
using Law.EdaIntegration.Ocr;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the MaxOcrAgents value
        edaCase.OCR.Properties.MaxOcrAgents = 4;
        // Save the changes to the case
        edaCase.OCR.UpdateProperties();

        // Get the value
        Console.WriteLine("MaxOcrAgents: {0}", edaCase.OCR.Properties.MaxOcrAgents);
    }
}
/*
This example produces the following results:

MaxOcrAgents: 4
 */
```

Remarks

A value of 0 represents an unlimited number of agents.

Exceptions

EdaApiException

An exception is thrown if the max OCR agents value is not between 0 and 999

MaxOcrableTextSize

Gets or sets the number of extracted text characters beyond which PDFs will not be OCR'ed, when the QueueForOcrDuringAnalysis and LimitOcrTextSize properties are both active.

```
int MaxOcrableTextSize { get; set; }
```

Property Value

int

Examples

The following example demonstrates setting this property.

```
using Law.EdaIntegration;
using Law.EdaIntegration.Case;
using Law.EdaIntegration.Ocr;
```

```
class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set LimitOcrTextSize and MaxOcrableTextSize
        edaCase.OCR.Properties.LimitOcrTextSize = true;
        edaCase.OCR.Properties.MaxOcrableTextSize = 200

        // Save the changes to the case
        edaCase.OCR.UpdateProperties();
    }
}
```

Remarks

Since we're not doing any parsing/limiting of this value in Explore, other than verifying that the value is a valid integer, we likewise won't add any restrictions here.

QueueForOcrDuringAnalysis

Indicates whether documents will be automatically queued for OCR during document analysis.

```
bool QueueForOcrDuringAnalysis { get; set; }
```

Property Value

bool

Examples

This example demonstrates how to set and retrieve the QueueForOcrDuringAnalysis value.

```
using Law.EdaIntegration;
using Law.EdaIntegration.Case;
using Law.EdaIntegration.Ocr;

class Sample
{
    public static void Main()
    {
        EdaIntegration edaIntegration = EdaIntegration.ConnectToExplore();
        Case edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Set the QueueForOcrDuringAnalysis value
        edaCase.OCR.Properties.QueueForOcrDuringAnalysis = true;

        // Save the changes to the case
        edaCase.OCR.UpdateProperties();

        // Get the value
        Console.WriteLine("QueueForOcrDuringAnalysis: {0}",
edaCase.OCR.Properties.QueueForOcrDuringAnalysis);
    }
}
/*
This example produces the following results:

QueueForOcrDuringAnalysis: True
 */
```

Remarks

Use AnalysisOcrFileTypes to specify which file types will be OCR'd when this property is enabled.

See Also

UpdateProperties()

Class InvalidOcrEngineException

Namespace: EdaIntegrationContract.OCR

Assembly: EdaIntegration.Contract.dll

Exception generated from the Integration Library when an invalid OCR engine is requested.

```
public class InvalidOcrEngineException : Exception, ISerializable
```

Inheritance

object ← Exception ← InvalidOcrEngineException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

InvalidOcrEngineException(string)

Exception generated from the Integration Library when an invalid OCR engine is requested.

```
public InvalidOcrEngineException(string message)
```

Parameters

`message` string

A description of the error

InvalidOcrEngineException(string, Exception)

Exception generated from the Integration Library when an invalid OCR engine is requested.

```
public InvalidOcrEngineException(string message, Exception innerException)
```

Parameters

`message` string

A description of the error

`innerException` Exception

The original exception that caused the error

Enum OcrEngineType

Namespace: EdaIntegrationContract.OCR

Assembly: EdaIntegration.Contract.dll

The OCR engines recognized by the Integration Library

```
public enum OcrEngineType
```

Fields

`Abbyy = 0`

Abby FineReader

`Expervision = 1`

ExperVision OpenRTK

`Vintasoft = 2`

VintaSoft Imaging

Namespace EdaIntegrationContract.OCR.Abbyy

Interfaces

IAbbyyOcrEngine

Abbyy FineReader optical character recognition engine

Enums

AbbyyLanguage

Specifies the languages supported by Abbyy FineReader

AbbyyPageLayout

Page layout settings supported by Abbyy FineReader

AbbyyPageQuality

The document page qualities supported by Abbyy FineReader

Enum AbbyyLanguage

Namespace: EdaIntegrationContract.OCR.Abbyy

Assembly: EdaIntegration.Contract.dll

Specifies the languages supported by Abbyy FineReader

```
public enum AbbyyLanguage
```

Fields

`ChinesePRC = 2052`

`ChineseTaiwan = 1028`

`Czech = 1029`

`Danish = 1030`

`Dutch = 1043`

`English = 1033`

`Finnish = 1035`

`French = 1036`

`German = 1031`

`Greek = 1032`

`Hebrew = 1037`

`Hungarian = 1038`

`Italian = 1040`

`Japanese = 1041`

`Korean = 1042`

`Latin = 1540`

```
Norwegian = 1044

Polish = 1045

PortugueseBrazilian = 1046

PortugueseStandard = 2070

Romanian = 1048

Russian = 1049

Spanish = 1034

Swedish = 1053

SystemDefault = 0

Thai = 1054
```

Enum AbbyyPageLayout

Namespace: EdaIntegrationContract.OCR.Abbyy

Assembly: EdaIntegration.Contract.dll

Page layout settings supported by Abbyy FineReader

```
public enum AbbyyPageLayout
```

Fields

`AutoDetect = 1`

Determine the layout automatically

`SingleColumn = 2`

Specifies only one column of text. Required before running Email Thread Analysis

Enum AbbyyPageQuality

Namespace: EdaIntegrationContract.OCR.Abbyy

Assembly: EdaIntegration.Contract.dll

The document page qualities supported by Abbyy FineReader

```
public enum AbbyyPageQuality
```

Fields

`DotMatrix = 3`

Select for pages printed via dot matrix printers, like the receipts from cash registers or ATMs

`Micr = 6`

Select for pages printed via Magnetic Ink Character Recognition (MICR) technology, like the routing numbers on checks

`Normal = 1`

Select for pages printed via inkjet printers, laser printers, or offset lithography

`OCR_A = 4`

Select for pages with text printed in OCR-A, which is a monospaced font designed specifically for OCR, and used on credit/debit cards

`OCR_B = 5`

Select for pages with text printed in OCR-B, which is another font designed specifically for OCR

`TypeWriter = 2`

Select for pages written via typewriter

Interface IAbbyyOcrEngine

Namespace: EdaIntegrationContract.OCR.Abbyy

Assembly: EdaIntegration.Contract.dll

Abbyy FineReader optical character recognition engine

```
public interface IAbbyyOcrEngine : IOcrEngine
```

Inherited Members

IOcrEngine.EngineType , IOcrEngine.Installed , IOcrEngine.IsActiveEngine

Properties

AutoDeskew

Specifies whether document page images are automatically de-skewed before OCR occurs

```
bool AutoDeskew { get; set; }
```

Property Value

bool

AutoRotate

Specifies when the OCR Engine should rotate document page images for the OCR output

```
bool AutoRotate { get; set; }
```

Property Value

bool

Language

Specifies a language dictionary that should be used by the OCR Engine

```
AbbyyLanguage Language { get; set; }
```

Property Value

AbbyyLanguage

PageLayout

Specifies the column layout of document pages

```
AbbyyPageLayout PageLayout { get; set; }
```

Property Value

AbbyyPageLayout

PageQuality

Specifies the printing technology used to create documents pages, as well as the quality of scanned pages

```
AbbyyPageQuality PageQuality { get; set; }
```

Property Value

AbbyyPageQuality

Namespace EdaIntegrationContract.OCR.Expervision

Interfaces

IExpervisionOcrEngine

ExperVision OpenRTK optical character recognition engine

Enums

ExpervisionAutoRotateSetting

Auto rotate settings supported by ExperVision OpenRTK

ExpervisionLanguage

Specifies the languages supported by ExperVision OpenRTK

ExpervisionPageLayout

Page layout settings supported by ExperVision OpenRTK

ExpervisionPageQuality

The document page qualities supported by ExperVision OpenRTK

Enum ExpervisionAutoRotateSetting

Namespace: EdaIntegrationContract.OCR.Expervision

Assembly: EdaIntegration.Contract.dll

Auto rotate settings supported by ExperVision OpenRTK

```
public enum ExpervisionAutoRotateSetting
```

Fields

`AlwaysOff = 1`

Images are never rotated

`AlwaysOn = 0`

Images are automatically rotated when needed

`BinaryImagesOnly = 2`

Only monochrome page images will be automatically rotated when needed

Enum ExpervisionLanguage

Namespace: EdaIntegrationContract.OCR.Expervision

Assembly: EdaIntegration.Contract.dll

Specifies the languages supported by ExperVision OpenRTK

```
public enum ExpervisionLanguage
```

Fields

`Danish = 64`

`Dutch = 128`

`English = 1`

`French = 2`

`German = 4`

`Italian = 8`

`Norwegian = 256`

`Portuguese = 32`

`Spanish = 16`

`Swedish = 512`

Enum ExpervisionPageLayout

Namespace: EdaIntegrationContract.OCR.Expervision

Assembly: EdaIntegration.Contract.dll

Page layout settings supported by ExperVision OpenRTK

```
public enum ExpervisionPageLayout
```

Fields

`AutoDetect = 1`

Determines the layout automatically

`SingleColumn = 2`

Specifies only one column of text. Required before running Email Thread Analysis

Enum ExpervisionPageQuality

Namespace: EdaIntegrationContract.OCR.Expervision

Assembly: EdaIntegration.Contract.dll

The document page qualities supported by ExperVision OpenRTK

```
public enum ExpervisionPageQuality
```

Fields

`DotMatrix = 68`

Select for pages printed via dot matrix printers, like the receipts from cash registers or ATMs for example

`DotMatrix_Degraded = 132`

Same as Dot Matrix, but the printed pages are of poor quality due to defective printing, photocopying, heavy use, or aging

`Normal = 67`

Select for pages printed via inkjet printers, laser printers, or offset lithography

`Normal_Degraded = 131`

Same as Normal, but the printed pages are of poor quality due to defective printing, photocopying, heavy use, or aging

Interface IExpervisionOcrEngine

Namespace: EdaIntegrationContract.OCR.Expervision

Assembly: EdaIntegration.Contract.dll

ExperVision OpenRTK optical character recognition engine

```
public interface IExpervisionOcrEngine : IOcrEngine
```

Inherited Members

IOcrEngine.EngineType , IOcrEngine.Installed , IOcrEngine.IsActiveEngine

Properties

AutoDeskew

Specifies whether document page images are automatically de-skewed before OCR occurs

```
bool AutoDeskew { get; set; }
```

Property Value

bool

AutoRotate

Specifies when the OCR Engine should rotate document page images for the OCR output

```
ExpervisionAutoRotateSetting AutoRotate { get; set; }
```

Property Value

ExpervisionAutoRotateSetting

Language

Specifies a language dictionary that should be used by the OCR Engine

```
ExpervisionLanguage Language { get; set; }
```

Property Value

ExpervisionLanguage

PageLayout

Specifies the column layout of document pages

```
ExpervisionPageLayout PageLayout { get; set; }
```

Property Value

ExpervisionPageLayout

PageQuality

Specifies the printing technology used to create documents pages, as well as the quality of scanned pages

```
ExpervisionPageQuality PageQuality { get; set; }
```

Property Value

ExpervisionPageQuality

Namespace EdaIntegrationContract.OCR.Vintasoft

Interfaces

IVintasoftOcrEngine

Vintasoft optical character recognition engine

Enums

VintasoftLanguage

Specifies the languages supported by Vintasoft Imaging

Interface IVintasoftOcrEngine

Namespace: EdaIntegrationContract.OCR.Vintasoft

Assembly: EdaIntegration.Contract.dll

Vintasoft optical character recognition engine

```
public interface IVintasoftOcrEngine : IOcrEngine
```

Inherited Members

IOcrEngine.EngineType , IOcrEngine.Installed , IOcrEngine.IsActiveEngine

Properties

AutoDeskew

Specifies whether document page images are automatically de-skewed before OCR occurs

```
bool AutoDeskew { get; set; }
```

Property Value

bool

AutoInvert

Indicates whether white text on a black background should also be recognized in the image

```
bool AutoInvert { get; set; }
```

Property Value

bool

AutoRotate

Specifies when the OCR Engine should rotate document page images for the OCR output

```
bool AutoRotate { get; set; }
```

Property Value

bool

ClearBorder

Indicates whether to identify and remove the black border that is sometimes produced when scanning images of documents

```
bool ClearBorder { get; set; }
```

Property Value

bool

Despeckle

Specifies whether to identify and remove black points due to dust deposited on the scanner capture components

```
bool Despeckle { get; set; }
```

Property Value

bool

HalfToneRemoval

Specifies whether to remove halftone dots when recognizing characters

```
bool HalfToneRemoval { get; set; }
```

Property Value

bool

HolePunchRemoval

Specifies whether to eliminate the holes scanner capture components

```
bool HolePunchRemoval { get; set; }
```

Property Value

bool

Language

Specifies a language dictionary that should be used by the OCR Engine

```
VintasoftLanguage Language { get; set; }
```

Property Value

VintasoftLanguage

Enum VintasoftLanguage

Namespace: EdaIntegrationContract.OCR.Vintasoft

Assembly: EdaIntegration.Contract.dll

Specifies the languages supported by Vintasoft Imaging

```
public enum VintasoftLanguage
```

Fields

`English = 1`

`Spanish = 2`

Namespace EdaIntegrationContract.Process

Interfaces

IAgentInfo

Status info for agents - proxy for EdaAgentInfo from Eda library trying to keep the eda library out of Turbo Import as much as possible

IProcessInfo

Provides Process information for the selected case.

Enums

AnalysisStates

Possible values for analysis state

InventoryStates

Possible values for inventory state

ProcessingStatus

The types of processing statuses

SourceStates

Possible values for source state

Enum AnalysisStates

Namespace: EdaIntegrationContract.Process

Assembly: EdaIntegration.Contract.dll

Possible values for analysis state

```
public enum AnalysisStates
```

Fields

`Complete = 3`

Analysis completed [Source.Analyzed >= Source.Inventoried]

`InProgress = 2`

An enqueued item has been processed [Source.Analyzed >1]

`None = 0`

Indicates no items have been enqueud and no analysis has been run. [Initial State]

`Pending = 1`

Indicates items have been enqueued, but no analysis has been run

Interface IAgentInfo

Namespace: EdaIntegrationContract.Process

Assembly: EdaIntegration.Contract.dll

Status info for agents - proxy for EdaAgentInfo from Eda library trying to keep the eda library out of Turbo Import as much as possible

```
public interface IAgentInfo : IEdaIntegrationEntity
```

Properties

Activity

Activity

```
string Activity { get; set; }
```

Property Value

string

Agent

AgentUuid

```
string Agent { get; set; }
```

Property Value

string

Attempts

RetryCount

```
int Attempts { get; set; }
```

Property Value

int

CaseUuid

CaseUuid

```
string CaseUuid { get; set; }
```

Property Value

string

FileName

FileName

```
string FileName { get; set; }
```

Property Value

string

FileSize

FileSize

```
long FileSize { get; set; }
```

Property Value

long

Machine

ServiceUuid

```
string Machine { get; set; }
```

Property Value

string

StartTime

StartTime

```
DateTime StartTime { get; set; }
```

Property Value

DateTime

Interface IProcessInfo

Namespace: EdaIntegrationContract.Process

Assembly: EdaIntegration.Contract.dll

Provides Process information for the selected case.

```
public interface IProcessInfo
```

Properties

Activity

Current Activity

```
string Activity { get; }
```

Property Value

string

RunningCount

Current Activity Count

```
int RunningCount { get; set; }
```

Property Value

int

ServiceAgentList

List of Service Agent by Activity

```
List<string> ServiceAgentList { get; set; }
```

Property Value

List<string>

Status

Current Status

```
ProcessingStatus Status { get; }
```

Property Value

ProcessingStatus

TimeRemaining

Time remaining to complete the Current Activity

```
string TimeRemaining { get; set; }
```

Property Value

string

Enum InventoryStates

Namespace: EdaIntegrationContract.Process

Assembly: EdaIntegration.Contract.dll

Possible values for inventory state

```
public enum InventoryStates
```

Fields

`Aborted = 3`

Indicates inventory abort/failure

`Complete = 4`

Indicates that the inventory has been completed

`InProgress = 2`

Indicates that inventory has been started [Set manually]

`Pending = 1`

Indicates the source has been configured and enqueued and is awaiting inventory

`Unconfigured = 0`

Source has yet to be configured and enqueued. [Initial State]

Enum ProcessingStatus

Namespace: EdaIntegrationContract.Process

Assembly: EdaIntegration.Contract.dll

The types of processing statuses

```
public enum ProcessingStatus
```

Fields

`Blacklisted = 4`

Processing on the case was suspended by the system

`Complete = 2`

No processing to be done

`Pending = 1`

Activities in queue

`Suspended = 3`

Processing on the case suspended by a user

`Working = 0`

Currently working on activities

Enum SourceStates

Namespace: EdaIntegrationContract.Process

Assembly: EdaIntegration.Contract.dll

Possible values for source state

```
public enum SourceStates
```

Fields

`Complete = 3`

Source processing completed

`DeleteFailed = 5`

Indicates the source delete failed

`DeletePending = 4`

Indicates the source has been queued for delete

`Failed = 2`

Source processing failed

`InProgress = 1`

An item in this source has been processed

`Pending = 0`

Indicates source has been added, but no work has been started

Namespace EdaIntegrationContract.Statistics

Interfaces

ICaseStatistics

Provides case level aggregate statistics.

IImportSetStatistics

Provides Import Set aggregate statistics.

ISourceStatistics

Provides processing statistics about the source.

IStatisticsCommon

Base statistics that are available

IStatisticsManager

Responsible for retrieving statistics about a case

Interface ICaseStatistics

Namespace: EdaIntegrationContract.Statistics

Assembly: EdaIntegration.Contract.dll

Provides case level aggregate statistics.

```
public interface ICaseStatistics : IStatisticsCommon
```

Inherited Members

IStatisticsCommon.AnalysisYield , IStatisticsCommon.AnalysisYieldSize , IStatisticsCommon.Errors , IStatisticsCommon.Exported , IStatisticsCommon.Failures , IStatisticsCommon.Inventoried , IStatisticsCommon.InventoriedSize , IStatisticsCommon.Warnings

Properties

ExcludedDupe

Total count of items that were excluded because they were duplicates.

```
int ExcludedDupe { get; }
```

Property Value

int

ExcludedDupeSize

Total size of items that were excluded because they were duplicates.

```
long ExcludedDupeSize { get; }
```

Property Value

long

ExcludedFtm

Total count of items that were excluded due to File Type database filter definitions.

```
int ExcludedFtm { get; }
```

Property Value

int

ExcludedFtmSize

Total size of items that were excluded due to File Type database filter definitions.

```
long ExcludedFtmSize { get; }
```

Property Value

long

ExcludedNist

Total count of items that were excluded due to Nist definitions.

```
int ExcludedNist { get; }
```

Property Value

int

ExcludedNistSize

Total size of items that were excluded due to Nist definitions.

```
long ExcludedNistSize { get; }
```

Property Value

long

ExportedToLaw

Total number of items exported to LAW.

```
int ExportedToLaw { get; }
```

Property Value

int

ExtractedFiles

Number of files extracted from the source data.

```
int ExtractedFiles { get; }
```

Property Value

int

FilesToOcr

Number of files that have been queued for OCR.

```
int FilesToOcr { get; }
```

Property Value

int

FilesWithOcr

Number of files marked in InventoryItems as having been OCR'ed (or having OCR attempted.)

```
int FilesWithOcr { get; }
```

Property Value

int

NumImportSets

Number of Import Sets in the case

```
int NumImportSets { get; }
```

Property Value

int

NumSources

Number of sources in the case

```
int NumSources { get; }
```

Property Value

int

Interface IImportSetStatistics

Namespace: EdaIntegrationContract.Statistics

Assembly: EdaIntegration.Contract.dll

Provides Import Set aggregate statistics.

```
public interface IImportSetStatistics : IStatisticsCommon
```

Inherited Members
IStatisticsCommon.AnalysisYield , IStatisticsCommon.AnalysisYieldSize , IStatisticsCommon.Errors , IStatisticsCommon.Exported , IStatisticsCommon.Failures , IStatisticsCommon.Inventoried , IStatisticsCommon.InventoriedSize , IStatisticsCommon.Warnings

Properties

Duplicates

The number of documents that have been excluded due to the Duplicate filter

```
int Duplicates { get; }
```

Property Value

int

DuplicatesSize

The size of documents that have been excluded due to the Duplicate filter

```
long DuplicatesSize { get; }
```

Property Value

long

ExportErrors

The number of errors encountered while exporting

```
int ExportErrors { get; }
```

Property Value

int

ExportedSize

The cumulative size of the documents that have been exported

```
long ExportedSize { get; }
```

Property Value

long

FileTypeExclusions

The number of documents that have been excluded due to the File Type filter

```
int FileTypeExclusions { get; }
```

Property Value

int

FileTypeExclusionsSize

The size of documents that have been excluded due to the File Type filter

```
long FileTypeExclusionsSize { get; }
```

Property Value

long

FilterExclusions

The number of documents that have been excluded due to one or more filters

```
int FilterExclusions { get; }
```

Property Value

int

FilterExclusionsSize

The size of documents that have been excluded due to one or more filters

```
long FilterExclusionsSize { get; }
```

Property Value

long

ImportSet

The ImportSet

```
IImportSet ImportSet { get; }
```

Property Value

IImportSet

Nisted

The number of documents that have been excluded due to the NIST filter

```
int Nisted { get; }
```

Property Value

[int](#)

NistedSize

The size of documents that have been excluded due to the NIST filter

```
long NistedSize { get; }
```

Property Value

[long](#)

PreFilterCount

The total number of top-level documents in the source before filtering is applied

```
int PreFilterCount { get; }
```

Property Value

[int](#)

PreFilterFamilyCount

The total number of documents and attachments, not including any containers, before filtering is applied

```
int PreFilterFamilyCount { get; }
```

Property Value

int

PreFilterFamilySize

The cumulative size of the documents and attachments, not including containers, before filtering is applied

```
long PreFilterFamilySize { get; }
```

Property Value

long

PreFilterSize

The cumulative size of all the top-level documents in the source before filtering is applied

```
long PreFilterSize { get; }
```

Property Value

long

Interface ISourceStatistics

Namespace: EdaIntegrationContract.Statistics

Assembly: EdaIntegration.Contract.dll

Provides processing statistics about the source.

```
public interface ISourceStatistics : IStatisticsCommon
```

Inherited Members
IStatisticsCommon.AnalysisYield , IStatisticsCommon.AnalysisYieldSize , IStatisticsCommon.Errors , IStatisticsCommon.Exported , IStatisticsCommon.Failures , IStatisticsCommon.Inventoried , IStatisticsCommon.InventoriedSize , IStatisticsCommon.Warnings

Properties

Duplicates

The number of documents that have been excluded due to the Duplicate filter

```
int Duplicates { get; }
```

Property Value

int

DuplicatesSize

The size of documents that have been excluded due to the Duplicate filter

```
long DuplicatesSize { get; }
```

Property Value

long

ExportErrors

The number of errors encountered while exporting

```
int ExportErrors { get; }
```

Property Value

int

ExportedSize

The cumulative size of the documents that have been exported

```
long ExportedSize { get; }
```

Property Value

long

FileTypeExclusions

The number of documents that have been excluded due to the File Type filter

```
int FileTypeExclusions { get; }
```

Property Value

int

FileTypeExclusionsSize

The size of documents that have been excluded due to the File Type filter

```
long FileTypeExclusionsSize { get; }
```

Property Value

long

FilterExclusions

The number of documents that have been excluded due to one or more filters

```
int FilterExclusions { get; }
```

Property Value

int

FilterExclusionsSize

The size of documents that have been excluded due to one or more filters

```
long FilterExclusionsSize { get; }
```

Property Value

long

ImportSetId

The identifier of the import set in which this source was imported.

```
int ImportSetId { get; }
```

Property Value

int

Nisted

The number of documents that have been excluded due to the NIST filter

```
int Nisted { get; }
```

Property Value

int

NistedSize

The size of documents that have been excluded due to the NIST filter

```
long NistedSize { get; }
```

Property Value

long

PreFilterCount

The total number of top-level documents in the source before filtering is applied

```
int PreFilterCount { get; }
```

Property Value

int

PreFilterFamilyCount

The total number of documents and attachments, not including any containers, before filtering is applied

```
int PreFilterFamilyCount { get; }
```

Property Value

int

PreFilterFamilySize

The cumulative size of the documents and attachments, not including containers, before filtering is applied

```
long PreFilterFamilySize { get; }
```

Property Value

long

PreFilterSize

The cumulative size of all the top-level documents in the source before filtering is applied

```
long PreFilterSize { get; }
```

Property Value

long

SourceId

The unique identifier for the source

```
int SourceId { get; }
```

Property Value

int

Interface IStatisticsCommon

Namespace: EdaIntegrationContract.Statistics

Assembly: EdaIntegration.Contract.dll

Base statistics that are available

```
public interface IStatisticsCommon
```

Properties

AnalysisYield

Total number of items processed.

```
int AnalysisYield { get; }
```

Property Value

int

AnalysisYieldSize

Total size of items processed.

```
long AnalysisYieldSize { get; }
```

Property Value

long

Errors

Total count of errors in processing and export.

```
int Errors { get; }
```

Property Value

int

Exported

Total number of items exported.

```
int Exported { get; }
```

Property Value

int

Failures

Total number of failures in processing.

```
int Failures { get; }
```

Property Value

int

Inventoried

Total number of items inventoried.

```
int Inventoried { get; }
```

Property Value

int

InventoriedSize

Total size of items inventoried.

```
long InventoriedSize { get; }
```

Property Value

long

Warnings

Total count of warnings in processing.

```
int Warnings { get; }
```

Property Value

int

Interface IStatisticsManager

Namespace: EdaIntegrationContract.Statistics

Assembly: EdaIntegration.Contract.dll

Responsible for retrieving statistics about a case

```
public interface IStatisticsManager
```

Methods

CaseStatistics()

Obtains the statistics for the case.

```
ICaseStatistics CaseStatistics()
```

Returns

ICaseStatistics

A ICaseStatistics containing the accumulated statistics for the case.

ImportSetSetStatistics(IEnumerable<int>)

Obtains the statistics for the provided list of Import Sets, or all Import Sets if none are provided.

```
IEnumerable<IImportSetStatistics> ImportSetSetStatistics(IEnumerable<int> importSetIds
= null)
```

Parameters

`importSetIds` IEnumerable<int>

A list of import set identifiers

Returns

IEnumerable<IImportSetStatistics>

A list if IImportSetStatistics containing the available statistics for each Import Set.

SourceStatistics(IEnumerable<int>)

Obtains the statistics for the provided list of sources, or all sources if none are provided.

```
IEnumerable<ISourceStatistics> SourceStatistics(IEnumerable<int> sourceIds = null)
```

Parameters

`sourceIds` IEnumerable<int>

A list of sources

Returns

IEnumerable<ISourceStatistics>

A list if ISourceStatistics containing the available statistics for each source.

Namespace EdaIntegrationContract.Tagging

Interfaces

ITag

Represents a tag item.

ITagManager

Represents a class for managing Tags.

Interface ITag

Namespace: EdaIntegrationContract.Tagging

Assembly: EdaIntegration.Contract.dll

Represents a tag item.

```
public interface ITag
```

Properties

Id

The unique Id of the tag.

```
int Id { get; }
```

Property Value

int

Name

The name of the tag.

```
string Name { get; set; }
```

Property Value

string

Remarks

Tag names can be a maximum of 50 characters. The name will be truncated if the supplied value is longer than the maximum length.

Interface ITagManager

Namespace: EdaIntegrationContract.Tagging

Assembly: EdaIntegration.Contract.dll

Represents a class for managing Tags.

```
public interface ITagManager
```

Methods

All()

Retrieves a list of ITag for a case.

```
IEnumerable<ITag> All()
```

Returns

IEnumerable<ITag>

A list of ITag items.

Examples

The following example demonstrates retrieving a list of tags for a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Tagging;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create tags
        edaCase.Tags.Create("Privileged");
        edaCase.Tags.Create("Responsive");

        foreach (ITag tag in edaCase.Tags.All())
        {
            Console.WriteLine("Id: "   + tag.Id);
            Console.WriteLine("Name: " + tag.Name);
        }
    }
}
/*
This example produces the following results:

Id: 1
Name: Privileged
Id: 2
Name: Responsive
 */
```

ById(int)

Retrieves a tag by its unique identifier.

```
ITag ById(int id)
```

Parameters

`id` int

The unique identifier of the tag to find.

Returns

ITag

The tag with the supplied unique identifier.

Examples

The following example demonstrates retrieving a tag using its id.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Tagging;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create a new tag
        ITag tag = edaCase.Tags.Create("Privileged");
        ITag itemById = edaCase.Tags.ById(tag.Id);
        Console.WriteLine("Id: {0}", itemById.Id);
        Console.WriteLine("Name: {0}", itemById.Name);
    }
}
/*
This example produces the following results:

Id: 1
Name: Privileged
 */
```

Exceptions

EdaApiException

Thrown if a tag cannot be found with the specified Id.

ByName(string)

Retrieves a tag by its name.

```
ITag ByName(string tagName)
```

Parameters

`tagName` string

The name of the tag to find.

Returns

ITag

The tag matching the supplied tag name.

Examples

The following example demonstrates retrieving a tag using its name.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Tagging;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create a new tag
        ITag tag = edaCase.Tags.Create("Privileged");
        ITag itemById = edaCase.Tags.ByName(tag.Name);
        Console.WriteLine("Id: {0}", itemById.Id);
        Console.WriteLine("Name: {0}", itemById.Name);
    }
}
/*
This example produces the following results:

Id: 1
Name: Privileged
 */
```

Exceptions

EdaApiException

Thrown if a tag cannot be found with the specified name.

Create(string)

Creates a new tag for a case.

```
ITag Create(string tagName)
```

Parameters

tagName string

The name of the tag to be created.

Returns

ITag

The tag object created.

Examples

The following example demonstrates how to create a new tag.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Tagging;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create a tag
        ITag tag = edaCase.Tags.Create("Privileged");
        Console.WriteLine("Id: {0}", tag.Id);
        Console.WriteLine("Name: {0}",tag.Name);
    }
}
/*
This example produces the following results:

Id: 1
Name: Privileged
 */
```

Remarks

Tags with duplicate names are not allowed to exist in a case.

Delete(IEnumerable<int>)

Delete one more tags from the case.

```
void Delete(IEnumerable<int> idsToDelete)
```

Parameters

`idsToDelete` IEnumerable<int>

The list of unique identifiers of the tags to be deleted.

Examples

This example demonstrates deleting a tag from a case.

```
using EdaIntegrationContract;
using EdaIntegrationContract.Case;
using EdaIntegrationContract.Tagging;

class Sample
{
    static void Main()
    {
        var edaIntegration = new EdaIntegration().ConnectToExplore();
        ICase edaCase = edaIntegration.Cases.OpenCaseByName("Case 1");

        // Create a tag
        ITag tag = edaCase.Tags.Create("Privileged");
        Console.WriteLine("Id: {0}", tag.Id);
        Console.WriteLine("Name: {0}",tag.Name);

        // Delete the tag
        List<int> itemsToDelete = new List<int> { tag.Id };
        edaCase.Tags.Delete(itemsToDelete);
        Console.WriteLine("Tag deleted");
    }
}
/*
```

```
This example produces the following results:

Id: 1
Name: Privileged
Tag deleted
 */
```