Quest®

Quest® InTrust 11.5

SDK Reference

If you have any questions regarding your potential use of this material, contact:

Quest Software Inc.

Attn: LEGAL Dept

4 Polaris Way

Aliso Viejo, CA 92656

Refer to our Web site (https://www.quest.com) for regional and international office information.

Patents

Quest Software is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at https://www.quest.com/legal.

Trademarks

Quest, the Quest logo, and Join the Innovation are trademarks and registered trademarks of Quest Software Inc. For a complete list of Quest marks, visit https://www.quest.com/legal/trademark-information.aspx. All other trademarks and registered trademarks are property of their respective owners.

Legend

CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.

IMPORTANT, **NOTE**, **TIP**, **MOBILE**, or **VIDEO**: An information icon indicates supporting information.

InTrust SDK Reference
Updated - September 2019
Version - 11.5

Contents

InTrust SDK Overview

The InTrust SDK makes InTrust functionality available to applications. At this time, the SDK includes the following components:

- Repository Services API
- Log Knowledge Base API

The InTrust SDK is included in the InTrust Server component and works on any computer where InTrust Server is deployed.

Requirements

If you want to install the SDK separately from InTrust Server, the computer must meet the following requirements (similar to the requirements for InTrust Server):

Architecture	x64
Operating System	Any of the following: • Microsoft Windows Server 2016 • Microsoft Windows Server 2012 R2 • Microsoft Windows Server 2012 • Microsoft Windows Server 2008 R2
Memory	Min. 6GB
Additional Software and Services	• Microsoft .NET Framework 4.5 with all the latest updates • Update for Universal C Runtime in Windows, available from https://support.microsoft.com/en-us/kb/2999226 at the time of this writing

CAUTION: To use the InTrust API with old versions of Windows PowerShell (2.0 and earlier), make sure you configure PowerShell to use the version of the .NET runtime that the SDK requires. For that, create the powershell.exe.config (or powershell_ise.exe.config) file in the same folder as powershell.exe (or powershell_ise.exe) file with content like the following:

```
<?xml version="1.0"?>
<configuration>
    <startup useLegacyV2RuntimeActivationPolicy="true">
        <supportedRuntime version="v4.0"/>
        <supportedRuntime version="v2.0.50727"/>
    </startup>
</configuration>
```

Required Permissions

To be able to use the features of the InTrust SDK, your code must be run under an account that is listed as an InTrust organization administrator. For details about setting up this privilege, see InTrust Organization Administrators.

Standalone Setup

To install the InTrust SDK separately from InTrust Server, run the **INTRUST_SDK.11.5.*.*.msi** installation package provided to you. It is located in the **InTrust\Server** folder in your InTrust distribution.

Configuring C# References

To make sure that C# bindings work, enable references to the following COM type libraries:

1. InTrust Environment 1.0 Type Library
2. Repository Record Inserter 1.0 Type Library
3. Repository Services 1.0 Type Library

For each of them, open the properties and set the **Embed Interop Types** parameter to **False**.

Repository Services API

This topic describes the API that InTrust provides for repositories. This API lets you do the following:

- Connect to a repository for searching and writing
- Get records from a repository by searching
- Put records in a repository
- Manage repositories:
    - Remove (unregister) them
    - Create them
    - Work with repository properties

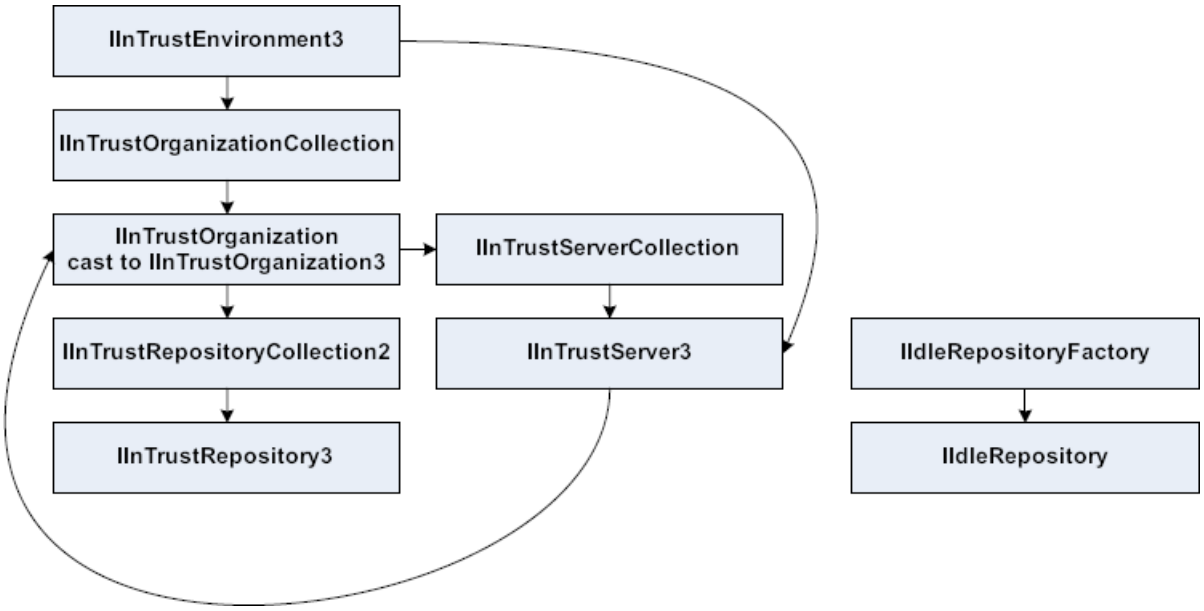
The API is implemented as a collection of COM objects that become available after you have installed the InTrust SDK. Use the interfaces described in the topics listed below; call the methods of those interfaces for access to records and repositories.

Connecting to a Repository

Use the interfaces listed below for access to an InTrust repository. Once you have gained access, you can search for records in the repository (see Getting Records) and write records to it (see Writing Records).

Overview of Repository Access

The following diagram shows the relationships between the InTrust SDK's interfaces used for getting access to a repository. An arrow indicates that an interface returns another interface.



Before you can have access to an InTrust repository, you need to initialize the InTrust environment. For that, create an object that implements the IInTrustEnvironment interface. This object makes the current InTrust organization, its servers and its repositories available to you. The relationships between these items are as follows:

- An InTrust organization provides a single configuration database for one or more InTrust servers.
- An InTrust repository is registered with an InTrust organization, and its entry is contained in the configuration shared by all InTrust servers in the organization.
- Specific InTrust servers manage specific repositories but do not “own” them; however, the organization does.

The IInTrustEnvironment interface provides the environment for working with all available InTrust organizations. You can use two methods to get the organization you need:

1. Get a collection of known organizations (**Organizations** method of the IInTrustEnvironment interface) and pick the necessary one. This involves working with the IInTrustOrganizationCollection interface. In this case, organizations are discovered by an Active Directory query.
2. Connect directly to an InTrust server by name (**ConnectToServer** method of the IInTrustEnvironment interface). This involves working with the IInTrustServer interface, which you can use to get the organization that the server is in.

Once you have gained access to an organization, use its interface (IInTrustOrganization3) to get a collection of the repositories in it (IInTrustRepositoryCollection2) and get the repository you are looking for (IInTrustRepository3).

The information above concerns access to regular production repositories. However, a valid file structure with data can also act as an InTrust repository for the purposes of searching and writing, even if it is not included in InTrust configuration. It is called an idle repository. An idle repository has no representation in the InTrust environment, so you need to construct its interface to gain access. For details, see Creating and Removing Repositories.

Examples

If you know the name of the organization for a specific repository, follow the **organization → repository** chain of access:

```
{
    IInTrustEnvironment intrust_environment = new InTrustEnvironment();
    IInTrustOrganizationCollection organizations = intrust_
environment.Organizations;
    IInTrustOrganization3 intrust_organization =
organizations.Cast<IInTrustOrganization3>().Where(x => x.Name == "My
Organization").First();
    IInTrustRepositoryCollection2 repositories = intrust_
organization.Repositories2;
    IInTrustRepository3 repository = repositories.Cast<IInTrustRepository3>
().Where(x => x.Name == "My Repository").First();
}
```

If you only know the name of a server in the organization, follow the **server → organization → repository** chain of access:

```
{
    IInTrustEnvironment intrust_environment = new InTrustEnvironment();
    IInTrustServer intrust_server = intrust_environment.ConnectToServer("My
Server");
    IInTrustOrganization3 intrust_organization = intrust_server.Organization as
IInTrustOrganization3;
    IInTrustRepositoryCollection2 repositories = intrust_organization.Repositories2;
    IInTrustRepository3 repository = repositories.Cast<IInTrustRepository3>().Where
(x => x.Name == "My Repository").First();
}
```

Details

Use the following interfaces for repository access and related tasks:

- IInTrustEnvironment
- IInTrustOrganizationCollection
- IInTrustOrganization
- IInTrustServerCollection
- IInTrustServer
- IInTrustRepositoryCollection2
- IInTrustRepository3

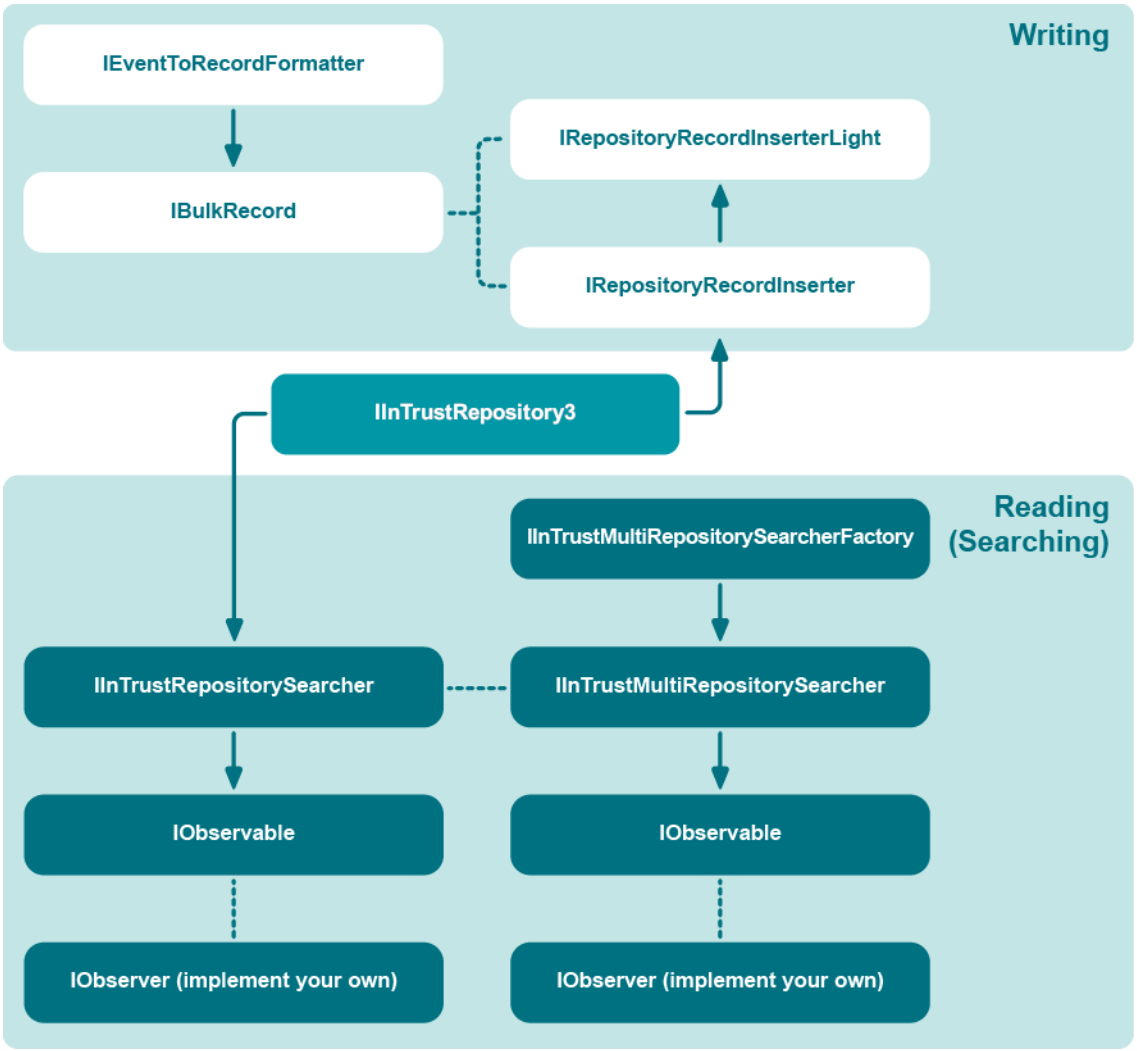
Getting and Putting Data

The InTrust repository was originally developed to store event log data, and this dictated the design choices that it is based on. However, the repository architecture is flexible enough for storing generic records containing arbitrary key-value pairs. The repository API provides tools for reading and writing both kinds of data.

Importantly, the repository is a document-oriented store. If you need to implement any inter-document relationships, you need to define them at the document contents level.

Overview

The following diagram shows the relationships between the InTrust SDK's interfaces used for reading and writing repository data. An arrow indicates that an interface returns another interface. Dashed lines between interfaces mean they don't return one another, but are used together for particular tasks.



See below for details about building program flow that uses these relationships. For a diagram of how to obtain the IInTrustRepository3 interface, see Connecting to a Repository.

Writing

Whether you want to write generic records or events, first you need access to the IRepositoryRecordInserter or IRepositoryRecordInserter2 interface. Take the following steps:

1. Connect to the repository you need, as described in Connecting to a Repository.
2. Get the IRepositoryRecordInserter or IRepositoryRecordInserter2 interface.
   This interface manages the writing of data to a repository. To obtain it, call the **Inserter** method of the IInTrustRepository3 interface of the repository you are connected to. A new inserter is created every time you make this call. You should obtain it once and reuse it for all writing to the repository.

For details about the next steps, see the following topics:

- Writing Records
- Writing Events

Reading

Reading data from a repository means searching the repository for it. Search queries use the REL language described in InTrust Customization Kit. For a list of fields that you can use in search queries, see Searchable Event and Record Fields. For some important REL query specifics, see Composing REL Queries.

The data-retrieving functionality of the InTrust repository API is modeled after the push-based notification system used in the Microsoft .NET Framework. Therefore, the API provides similar interfaces (such as IObservable and IObserver).

To perform a repository search

1. Connect to the repository you need, as described in Connecting to a Repository. This gives you access to the IInTrustRepository3 interface.
2. Use the **Searcher** method of the IInTrustRepository3 interface to get the IInTrustRepositorySearcher interface.
3. Use that interface's **Search** method to get an IObservable interface.
4. Subscribe to the notification using the IObserver interface.

Example of a helper function (C#):

```
static void search_events(IInTrustRepository intrust_repository, string
query)
{
    IObservable observable = intrust_repository.Searcher().Search(query);
    MyObserver observer = new MyObserver();
    observable.Subscribe(observer, out observer.m_cookie);
}
```

The repository API also provides a way to perform searches on multiple repositories simultaneously. The IMultiRepositorySearcher interface is provided for this purpose.

To perform a multi-repository search

1. Obtain the IInTrustRepositorySearcher interfaces for the repositories you need.
2. Construct a IMultiRepositorySearcher interface using the IMultiRepositorySearcherFactory interface.
3. In the newly-created interface, specify the interfaces from the first step using the **MakeMultiSearchObject** method.
4. Use the returned interface as a regular IInTrustRepositorySearcher interface, as described above.

Example of a multi-repository search:

```
IInTrustEnvironment env = new InTrustEnvironment();
IInTrustServer server = env.ConnectToServer("10.30.38.230");
IInTrustOrganization org = server.Organization;
IInTrustEventory evs = org.Eventory;
string eventory_str = evs.Eventory;
IMultiRepositorySearcherFactory multi_searcher_fac = new
MultiRepositorySearcherFactory();
IMultiRepositorySearcher multi_searcher = multi_searcher_
fac.CreateMultiRepositorySearcher(eventory_str);
```

The example above involves an explicitly specified log knowledge base (see Log Knowledge Base API for details). To use the default log knowledge base, rewrite it as follows:

```
IMultiRepositorySearcherFactory multi_searcher_fac = new
MultiRepositorySearcherFactory();
IMultiRepositorySearcher multi_searcher = multi_searcher_
fac.CreateMultiRepositorySearcher(null);
```

For details about the next steps, see the following topics:

- Getting Records
- Getting Events

The following interfaces are involved in repository searches:

- IObservable
  Enables push-based notification. Implement this interface as the source of discovered records.
- IObserver
  Gets push-based notifications. Implement this interface as the search result handler.
- ICookie
  Keeps a search active. It is unlikely that you will need to handle this interface directly, but it helps to know that it is involved in searching.

Getting Records

A repository search returns data wrapped in a polymorphic **IUnknown** interface, as described in Getting and Putting Data. To interpret the data as repository records, cast it as IBulkRecord2.

Example (C#)

```
class MyObserver : IDisposable, REPOSITORYSERVICESLib.IObserver
{
    public REPOSITORYSERVICESLib.ICookie m_cookie;
    public MyObserver()
    {
    }
    public void OnDone()
    {
        Console.WriteLine("Search done");
    }
    public void OnError(int hr, string description)
    {
        Console.WriteLine("Search error: {0}", description);
    }
    public void OnNext(object data)
    {
        if (data != null)
        {
            IBulkRecord2 bulk_record2 = (data as IBulkRecord2);
            List<record2> records = bulk_record.GetRecords().Cast<record2>
().ToList<record2>();
            int record_count = 0;
            foreach (record2 my_record in records)
            {
                ++record_count;
            }
        }
        System.Runtime.InteropServices.Marshal.FinalReleaseComObject(data);
    }
}
```

For details about what repository records are, see Repository Record Data Structures.

Getting Events

A repository search returns data wrapped in a polymorphic **IUnknown** interface, as described in Getting and Putting Data. To interpret the data as event records, cast it as IBulkEventWithReadExtensions.

Example (C#)

```
class MyObserver : IDisposable, REPOSITORYSERVICESLib.IObserver
{
    public REPOSITORYSERVICESLib.ICookie m_cookie;
    public MyObserver()
    {
    }
    public void OnDone()
    {
        Console.WriteLine("Search done");
```

```
    }
    public void OnError(int hr, string description)
    {
        Console.WriteLine("Search error: {0}", description);
    }
    public void OnNext(object data)
    {
        if (data != null)
        {
            IBulkEventWithReadExtensions bulk_event = (data as
IBulkEventWithReadExtensions);
            List<event_with_read_extensions> events = bulk_event.GetEvents
().Cast<event_with_read_extensions>().ToList<event_with_read_extensions>();
            int event_count = 0;
            foreach (event_with_read_extensions my_event in events)
            {
                ++event_count;
            }
        }
        System.Runtime.InteropServices.Marshal.FinalReleaseComObject(data);
    }
}
```

For details about what event records are, see Event Record Data Structures.

Composing REL Queries

REL is an expression language developed specifically for InTrust, and it is used for multiple purposes besides repository searching.

The following topics about REL in the InTrust Customization Kit contain information that is fully applicable to queries used for searching in repositories:

- Words
- Expressions
- Operators
- Functions

However, due to the specifics of how repositories operate, there are some limitations on what you can include in your queries and nuances that affect performance. These peculiarities have to do with the following:

- Use of punctuation in field values
- Whether "equals" or "contains" semantics are used

Punctuation and Other Non-Alphanumeric Characters

Some characters, such as curly braces and the hyphen, are treated in a special way by the repository indexing engine. A query that includes these characters is automatically transformed during an indexed search, even though the query itself may be perfectly valid. The indexing engine splits the query into substrings at these characters and uses the substrings to make the clauses of an AND expression.

As a result, these characters are effectively removed from the index. This affects how well irrelevant data is filtered out and, consequently, how fast queries are evaluated. The following is a list of such characters:

- \ & { } ( ) [ ] < > , ! ? .

You can deal with this limitation in the following ways:

What you can do	Comments
Do nothing; leave the characters where they are.	Your query will be transformed automatically so that the indexing engine filters out as much of the repository as it can before running the search. However, punctuation characters are not part of the index, so the expected values may not be the only ones that match. A lot of similar but irrelevant matches can be present in the results. How fast your query runs and how relevant its results are depends on how many distinctive alphanumeric substrings it contains besides the punctuation: • If your query is made up of strings that have low chances of occurrence, your search will be fast and the results will be mostly relevant. • If your query contains strings that occur all the time, your search will be slow and the results will not be very useful.
Pick different fields to match by; ones that can contain only alphanumeric characters.	You can achieve maximum search performance, but it can be difficult or impossible to find equally relevant fields.

"Equals" Versus "Contains"

In a repository search, a query that uses "equals" semantics (the **striequ** REL function) is always evaluated faster than a similar query using "contains" semantics (the **substr** REL function). Queries with "does not equal" semantics are even slower.

Regular expressions (the **regexp** REL function) are slowest.

Searchable Event and Record Fields

This topic lists the field names that you can use in your REL queries when you search for events or records in a repository using the IObservable and IObserver interfaces.

The results of a search are polymorphic and can be cast to events or records as necessary. In addition, you can treat the contents of the repository as either events or records and use either event field names or record field names. However, you cannot mix event and record field names in the same query.

For example, if your repository contains custom records with filled-in insertion strings, it is convenient to treat the records as events for easy access to insertion string contents (see Insertion Strings below).

Event Fields

Field	Details
__AnyField	Look for the specified pattern in all fields.
Category	A symbolic representation of the event category. Search pattern example: **"(\\b|\\W|^)security"**

Field	Details
Computer	The computer where the event was logged.
Environment	Internally, InTrust predefines two environment ID values: • **8EAF6C85-D1FF-4CFD-9D90-64944C8E6B3E** Unix Network Environment • **9E442BEE-EAC2-4D79-9013-053FB225CFD0** Microsoft Windows Network Environment Custom environment IDs can also occur.
EventID	
PlatformID	Internally, InTrust predefines the following platform ID values: • 500 Microsoft Windows • 610 Solaris • 620 HP-UX • 630 Linux • 640 IBM AIX Custom platform IDs can also occur.
Source	The subsystem or service that the event is related to. For example, in forwarded Syslog events the source is "Syslog Device".
SourceComputer	The computer where the event originated; this can be different from the computer where it was logged.
SourceDomain	The domain of the computer where the event originated, if applicable.
Time	The timestamp in the event. **Tip:** Use filtering by date in your REL queries whenever the date range is known. This speeds up searches considerably.
Type	The predefined types are **Information**, **Warning**, **Error**, **Failure Audit** and **Success Audit**.
UserDomain	The domain of the user who produced the event.
UserName	The name of the user who produced the event.
VersionMajor	The major operating system version number of the computer on which the event occurred. For example, the major version of Windows 7 is 6.
VersionMinor	The minor operating system version number of the computer on which the event occurred.

Field	Details
	For example, the minor version of Windows 7 is 1.
What	A brief description of what the event is about.
Where	The computer where the event happened (had effect).
Where_From	The name or IP address of the computer from which the activity (such as a logon or configuration change) was performed. This is not necessarily the same computer as the one where the activity had effect.
Who	The plain user name of the account that caused the event.
WhoDomain	The Active Directory domain of the account that caused the event, where applicable.
Whom	The user account that was affected by the event, where applicable.

Insertion Strings

To look in insertion strings and resolved insertion strings, respectively, use the following field names:

- InsertionString*N*
- ResolvedInsertionString*N*

where *N* is the number of the string.
Examples of REL expressions:

```
in( InsertionString10, "rei", "(\\b|\\W|^)is1608133597" );
```

```
striequ(ResolvedInsertionString2,"is");
```

Record Fields

Most of the fields defined in the record data structures (see Repository Record Data Structures) can be used in search queries:

- directory_tag_1
- directory_tag_2
- directory_tag_3
- directory_tag_4
- field_1
- field_3
- field_4
- file_tag_1
- file_tag_2
- file_tag_3
- file_tag_4
- formatting_record_field

- string_field_1
- string_field_2
- string_field_3
- string_field_4
- string_field_5

Note that some fields contain integers and others strings.

Examples of REL expressions:

```
field_1 = 123;

striequ(directory_tag_1,"blue");

striequ(formatting_record_field,"green");

striequ(string_field_1,"cerise");

file_tag_1 = 5385;
```

Writing Records

After you have obtained the IRepositoryRecordInserter or IRepositoryRecordInserter2 interface (as described in Getting and Putting Data), you need to generate the data structures that you are going to write. Before you begin writing, make sure you understand the record data structures (see Repository Record Data Structures) and are able to construct them efficiently. The repository API provides two ways to write records: you can use either complete **record** structures or arrays of the smaller structures from which **record**s are made up. The next steps depend on this choice.

Approach 1: Writing Whole Record Structures

In this approach, you combine your newly generated **tags** and **contents** structures into complete **record** structures, put the **record**s in an array and supply the array to the **PutRecords** method of the IRepositoryRecordInserter interface.

Approach 2: Splitting Records for Writing

This approach requires that you plan in advance which of your record fields best to use as **tags**. This will enable you to create records with shared **tags** and different **contents**. The IRepositoryRecordInserterLight interface stores the **tags** that you want to share among your records. To get this interface, call the **BindFields** method of the IRepositoryRecordInserter interface you have obtained. This method accepts your **tags** as a parameter.

After that, supply the **contents** parts of your **record**s to the IRepositoryRecordInserterLight interface; it will form complete **record** structures and perform the writing.

Example (C#)

```
using System;
```

```
using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

using REPOSITORYSERVICESLib;

using REPOSITORYRECORDINSERTERLib;

using INTRUSTENVIRONMENTLib;

namespace RepositoryRecordInserterTest2

{

    class Program

    {

        public static uint ToUnixTime(DateTime date)

        {

            var epoch = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);

            return (uint)Convert.ToInt64((date.ToUniversalTime() -
 epoch).TotalSeconds);

        }

        public static DateTime UnixTimestampToDateTime(uint unixTime)

        {

            DateTime unixStart = new DateTime
(1970, 1, 1, 0, 0, 0, 0, System.DateTimeKind.Utc);

            long unixTimeStampInTicks = (long)(unixTime * TimeSpan.TicksPerSecond);

            return new DateTime(unixStart.Ticks + unixTimeStampInTicks);

        }


        class MyObserver : IDisposable, REPOSITORYSERVICESLib.IObserver

        {

            private AutoResetEvent m_waitHandler;

            public int event_count;

            public REPOSITORYSERVICESLib.ICookie m_cookie;
```

```
        public MyObserver(AutoResetEvent x)
        {
            m_waitHandler = x;
            event_count = 0;
        }
        public void OnDone()
        {
            Console.WriteLine("Search done");
            m_waitHandler.Set();
        }
        public void OnError(int hr, string description)
        {
            Console.WriteLine("Search error - {0}", description);
            m_waitHandler.Set();
        }
        public void OnNext(object data)
        {
            if (data != null)
            {
                IBulkRecord bulk_event = (data as IBulkRecord);
                List<record> records = bulk_event.GetRecords().Cast<record>
().ToList<record>();
                foreach (record my_record in records)
                {
                    Console.WriteLine("next record");
                    Console.WriteLine("    time - {0}", UnixTimestampToDateTime
(my_record.record_contents.gmt_time));
                    foreach (named_string my_named_string in my_record.record_
contents.named_fields)
                    {
                        Console.WriteLine("    key - {0}, value - {1}", my_named_
string.name, my_named_string.value);
```

```
                    }
                    ++event_count;
                }
            }
            System.Runtime.InteropServices.Marshal.FinalReleaseComObject(data);
        }
        public void Dispose()
        {
            m_cookie.Stop();
        }
    }
    static tags construct_sharding_fields(string log)
    {
        // For details about using sharding keys, see the "Repository Record Data
Structures" topic
        tags tg = new tags();
        tg.directory_tag_1 = "ShardingLevel1";
        tg.directory_tag_2 = "ShardingLevel2";
        tg.directory_tag_3 = "{A9E5C7A2-5C01-41B7-9D36-E562DFDDEFA9}"; // Sharding
level 3 must be a GUID
        tg.directory_tag_4 = log;
        tg.file_tag_1 = 0;
        tg.file_tag_2 = 500;
        tg.file_tag_3 = 0;
        tg.file_tag_4 = 0;
        return tg;
    }
    static contents construct_contents_fields(insertion_string[] insertion_
strings, named_string[] named_fields, string formatting_record_field)
    {
        contents ct = new contents();
        ct.string_field_1 = "string_field_1_value";
```

```
            ct.string_field_2 = "string_field_2_value";
            ct.string_field_3 = "string_field_3_value";
            ct.string_field_4 = "string_field_4_value";
            ct.string_field_5 = "string_field_5_value";
            ct.formatting_record_field = formatting_record_field;
            ct.gmt_time = ToUnixTime(DateTime.Now);
            ct.field_1 = 300;
            ct.field_2 = 50;
            ct.field_3 = 2;
            ct.field_4 = 3;
            ct.strings = insertion_strings;

            ct.named_fields = named_fields;
            return ct;
        }
        static record construct_record(uint index, string logname, insertion_string
[] insertion_strings, named_string[] named_fields, string description)
        {

            return new record() {
                record_path = construct_sharding_fields(logname),
                record_contents = construct_contents_fields(insertion_strings, named_
fields, description)
            };

         }
        static void insert_records(IRepositoryRecordInserter pInserter)
        {
            DateTime start = DateTime.Now;
            List<record> records = new List<record>();
            for (uint i = 0; i != 16000; ++i)
            {
                insertion_string[] insertion_strings =
```

```
                    {
                        new insertion_string() { index = 1, value = "My" },
                        new insertion_string() { index = 2, value = "String
value 2" },
                        new insertion_string() { index = 6, value = "Event" },
                        new insertion_string() { index = 7, value = "String value 7" }
                    };
                named_string[] named_fields =
                    {
                        new named_string()
{ name = "FieldName1", value = "FieldValue1"},
                        new named_string()
{ name = "FieldName2", value = "FieldValue2"},
                        new named_string()
{ name = "FieldName3", value = "FieldValue3"},
                        new named_string()
{ name = "FieldName4", value = "FieldValue4"},
                        new named_string()
{ name = "FieldName5", value = "FieldValue5"},
                    };
                records.Add(construct_record(i, "Log1", insertion_strings, named_
fields, "This %1 %6 description"));
            }
            pInserter.PutRecords(records.ToArray());
            pInserter.Commit();
        }
        static void insert_records_on_server(IRepositoryRecordInserter2 pInserter)
        {
            DateTime start = DateTime.Now;
            List<record> records = new List<record>();
            for (uint i = 0; i != 16000; ++i)
            {
                insertion_string[] insertion_strings =
                    {
```

```
                        new insertion_string() { index = 1, value = "My" },
                        new insertion_string() { index = 2, value = "String
value 2" },
                        new insertion_string() { index = 6, value = "Event" },
                        new insertion_string() { index = 7, value = "String value 7" }
                    };
                named_string[] named_fields =
                    {
                        new named_string()
{ name = "FieldName1", value = "FieldValue1"},
                        new named_string()
{ name = "FieldName2", value = "FieldValue2"},
                        new named_string()
{ name = "FieldName3", value = "FieldValue3"},
                        new named_string()
{ name = "FieldName4", value = "FieldValue4"},
                        new named_string()
{ name = "FieldName5", value = "FieldValue5"},
                    };
                records.Add(construct_record(i, "Log1", insertion_strings, named_
fields, "This %1 %6 description"));
            }
            pInserter.PutRecords(records.ToArray());
            pInserter.Commit2(ToServerRepositoryCommitType);
        }
        static tags construct_naive_sharding_fields(string log)
        {
            // For details about using sharding keys, see the "Repository Record Data
Structures" topic
            tags tg = new tags();
            tg.directory_tag_1 = "ShardingLevel1";
            tg.directory_tag_2 = "ShardingLevel2";
            tg.directory_tag_3 = "{A9E5C7A2-5C01-41B7-9D36-E562DFDDEFA9}"; // Sharding
level 3 must be a GUID
            tg.directory_tag_4 = log; // ShardingLevel4
```

```
            return tg;
        }
        static contents construct_naive_contents_fields(named_string[] named_fields)
        {
            contents ct = new contents();
            ct.gmt_time = ToUnixTime(DateTime.Now);
            ct.named_fields = named_fields;
            return ct;
        }
        static record construct_naive_record(uint index, string logname, named_string
[] named_fields)
        {
            return new record()
            {
                record_path = construct_naive_sharding_fields(logname),
                record_contents = construct_naive_contents_fields(named_fields)
            };
        }
        static void insert_naive_records(IRepositoryRecordInserter pInserter)
        {
            DateTime start = DateTime.Now;
            List<record> records = new List<record>();
            for (uint i = 0; i != 16000; ++i)
            {
                named_string[] named_fields =
                    {
                        new named_string()
{ name = "FieldName1", value = "FieldValue1"},
                        new named_string()
{ name = "FieldName2", value = "FieldValue2"},
                        new named_string()
{ name = "FieldName3", value = "FieldValue3"},
```

```
                        new named_string()
{ name = "FieldName4", value = "FieldValue4"},

                        new named_string()
{ name = "FieldName5", value = "FieldValue5"},

                    };

                records.Add(construct_naive_record(i, "Log1", named_fields));

            }

            pInserter.PutRecords(records.ToArray());

            pInserter.Commit();

        }

        static void search_records(IInTrustRepository intrust_
repository, string query)

        {

            IObservable observable = intrust_repository.Searcher.Search(query);

            AutoResetEvent waitHandler = new AutoResetEvent(false);

            MyObserver observer = new MyObserver(waitHandler);

            observable.Subscribe(observer, out observer.m_cookie);

            waitHandler.WaitOne();

        }


        static void records_example(IInTrustRepository intrust_repository)

        {

            IRepositoryRecordInserter pInserter = intrust_repository.Inserter;

            // Insert records with strictly key-value data directly to the repository

            insert_naive_records(pInserter);

            search_records(intrust_repository, "(in( __AnyField, \"rei\", \"
(\\\\b|\\\\W|^)FieldValue5\" ));");

            // Insert records directly to the repository

            insert_records(pInserter);

            search_records(intrust_repository, "(in( __AnyField, \"rei\", \"
(\\\\b|\\\\W|^)String value 7\" ));"

            // Insert records by queuing them on the server

            insert_records_on_server(pInserter as IRepositoryRecordInserter2);
```

```
            System.Threading.ThreadSleep(60000);// We need to wait, because there will
be a delay up to a minute before the event arrives in the repository

            search_records(intrust_repository, "(in( __AnyField, \"rei\", \"
(\\\\b|\\\\W|^)String value 7\" ));"

        }

        static void Main(string[] args)

        {

            if (args.Length != 2)

            {

                Console.WriteLine("Invalid argument count.\n");

                Console.WriteLine("\tRepositoryRecordInserterTest.exe <InTrust Server
Binding String> <Repository Name>\n");

                return;

            }

            try

            {

                InTrustEnvironmentintrust_environment = new InTrustEnvironment();

                IInTrustOrganization3 intrust_organization = intrust_
environment.ConnectToServer(args[0]).Organization as IInTrustOrganization3;

                IInTrustRepository3 intrust_repository = intrust_
organization.Repositories2.Cast<IInTrustRepository3>().Where(x => x.Name == args
[1]).First();

                records_example(intrust_repository);

            }

            catch (Exception e)

            {

                Console.WriteLine("Error : {0}", e.ToString());

            }

        }

    }

}
```

Writing Events

After you have obtained the IRepositoryRecordInserter interface (as described in Getting and Putting Data), take the following steps:

1. Generate valid **event_with_extensions** structure instances; this structure is described in Event Record Data Structures.
2. Through combined use of the IEventToRecordFormatter and IRepositoryRecordInserter interfaces, supply your newly generated events for writing.
   For that, use the **Format** method of the IEventToRecordFormatter interface to wrap your events in IBulkEventWithReadExtensions, and pass the resulting wrapper interface as a parameter to the **PutRecords2** method of the IRepositoryRecordInserter interface.

Sort Order for Writing

When you write events in batches, the events must be sorted by **gmt_time**, but not necessarily throughout the entire batch. The important thing is to sort those events where the following are the same: domain, computer, log name. That is the scope where you need to sort. For example, if your event batch contains 1000 events from 1000 computers, no sorting is necessary. But if it is 1000 events from two computers, you need to do two sorts.

Out-of-Order Writing

Submitting events out of order is possible, but there is a serious caveat. Whenever the timestamp of your event is less than that of the event you submitted last, you must use the **Commit** method of your inserter interface before you write the “flashback” event. Here are some implications of this:

- Events that are newer than the last-submitted event and older than that event should not be put in the same batch. Otherwise, timestamps will not be interpreted correctly. For example, searching within a specific time range will not return events that match but were written out of order between commits.
- A huge amount of repository files (caused by frequent commits) is bad for performance. Batch your “old” events as much as possible.

Repository Record Data Structures

A record is a chunk of data that is (or has been prepared for being) stored in a repository and processed by repository searches. A record is made up of two parts: tags and contents. The tags indicate where in the file system the file with this record is located. At the same time, the tags double as record field values. The contents do not do anything other than contain record field values.

When you create records, it is important which fields you use as tags and which you use as contents. By handling tags rationally you can help speed up searches on the data you are storing and minimize disk usage by your repository contents.

When you search for records, it doesn't matter in search queries whether a value is used as a tag or as contents.

For recommendations on efficiently organizing data fields in records, see Recommendations on Setting Tags below.

tags

```
struct tags

{

    BSTR directory_tag_1;

    BSTR directory_tag_2;

    BSTR directory_tag_3; // must specify a GUID

    BSTR directory_tag_4;

    unsigned file_tag_1;

    unsigned file_tag_2;

    unsigned file_tag_3;

    unsigned file_tag_4;

};
```

Contains the values of eight of the record's fields. In addition to carrying record data, this combination of fields is used for identifying the path to a folder and the name of a file in the repository tree. Repository trees are four levels deep, and files are located only at the deepest level.

The directory tags specify the four nesting levels. The third level must be a string representation of a GUID; InTrust verifies the format. This requirement is due to the implementation of the repository services. The GUID is used for identifying event-providing data sources, and each data source type has a particular known GUID. You may want to come up with your own set of special-purpose GUIDs for your specific tasks (for example, hierarchical organization).

The file tags specify the four parts of the file's base name. A file represented by this structure contains one or more entries represented by **contents** and **contents2** structures. If the number of entries per file hits a limit, the same base name is used for creating a new file, and the entries are continued in it.

In a simplified way, the location of a repository file in the file system hierarchy can be represented as follows:

repository_root
└ directory_tag_1
  └ directory_tag_2
    └ directory_tag_3
      └ directory_tag_4
└ file_tag_1_file_tag_2_ file_tag_3_file_tag_4_InTrust_internal_tags

Because tags correlate with the file system, make sure their values do not contain characters that are disallowed in file and directory names.

contents

```
struct contents

{

    unsigned field_1;

    unsigned field_2;
```

```
    short field_3;
    short field_4;
    DATE gmt_time;
    BSTR string_field_1;
    BSTR string_field_2;
    BSTR string_field_3;
    BSTR string_field_4;
    BSTR string_field_5;
    SAFEARRAY(struct insertion_string) strings;
    SAFEARRAY(struct named_string) named_fields;
    BSTR formatting_record_field;
};
```

Used for writing records to the reposiotry and contains the remaining values of the record fields in addition to those specified by the **tags** structure. This structure is physically represented by an entry in a repository file.

NOTE: The InTrust repository is known to easily handle up to 300 strings per record. Higher numbers of strings have not been tested and cannot be recommended.

As a best practice, make sure that your string mapping is consistent; that is, the same string numbers should have the same meanings across your records. This is beneficial for repository searches.

CAUTION: At this time, the field_2 field is not indexed and cannot be processed in repository searches. Writing useful data to this field is currently not recommended.

contents2

```
struct contents2
{
    unsigned field_1;
    unsigned field_2;
    short field_3;
    short field_4;
    DATE gmt_time;
    BSTR string_field_1;
    BSTR string_field_2;
    BSTR string_field_3;
    BSTR string_field_4;
    BSTR string_field_5;
```

```
    SAFEARRAY(struct insertion_string) strings;

    SAFEARRAY(struct named_string) native_named_fields;

    SAFEARRAY(struct named_string) resolved_named_fields;

    BSTR formatting_record_field;

};
```

Used in results of repository searches and contains the remaining values of the record fields in addition to those specified by the **tags** structure. This structure is physically represented by an entry in a repository file.

Unlike the **contents** structure, the **contents2** structure contains two distinct arrays of **named_string** structures. This is because the data for the **native_named_fields** field is not known during record writing. It is only available to repository searches.

NOTE: The InTrust repository is known to easily handle up to 300 strings per record. Higher numbers of strings have not been tested and cannot be recommended.

As a best practice, make sure that your string mapping is consistent; that is, the same string numbers should have the same meanings across your records. This is beneficial for repository searches.

CAUTION: At this time, the field_2 field is not indexed and cannot be processed in repository searches. Writing useful data to this field is currently not recommended.

record

```
struct record

{

    struct tags record_path;

    struct contents record_contents;

};
```

Combines the path-specifying (**tags**) and complementary (**contents**) parts of a record into a single structure. This type of record is used for writing to the repository.

record2

```
struct record2

{

    struct tags record_path;

    struct contents2 record_contents;

};
```

Combines the path-specifying (**tags**) and complementary (**contents2**) parts of a record into a single structure. This type of record is returned by repository searches.

Recommendations on Setting Tags

Organize your record tags according to the likelihood of the value being the same in a given collection of records. That is, **directory_tag_1** should contain the value that is the same in most of the records you are about to generate. Conversely, **directory_tag_4** should contain the value that the fewest records have in common. Also note that the best way to map the tags (and thereby define how the records will be stored physically) is to make them correspond to something that falls into a meaningful hierarchy. For example, all your records might be Security log events, but it still doesn't make sense to make the log name the topmost level. You would do better to tag map **directory_tag_1**,..., **directory_tag_4** to domain, computer, data source and log name, respectively; even though there is more value variation at the top levels this way.

A repository can store heterogeneous objects, but you need a way to tell their types apart. This requires a generic ID field, and **directory_tag_3** is good for the purpose. If you come up with a GUID for each object type (file system, computer, Active Directory object and so on), you will not confuse them. A further improvement is to design a hierarchy of IDs.

Event Record Data Structures

These data structures are alternatives to generic repository record data structures. Use event records for convenience when your records represent log events. For details about the meaning of the fields used in event records, see Event Record Data Structures below.

The primary data structure is base_event. There are also two structures that extend it: event_with_extensions and event_with_read_extensions. For details about the use of these data structures, see Getting Events and Writing Events.

base_event

```
struct base_event

{

    BSTR environment;

    BSTR gathering_domain;

    BSTR gathering_computer;

    BSTR datasource_type;

    BSTR gathered_event_log;

    BSTR user_name;

    BSTR user_domain;

    BSTR source_name;

    BSTR computer_name;

    BSTR string_category;

    BSTR description_template;

    SAFEARRAY(struct insertion_string) strings;
```

```
    SAFEARRAY(unsigned char) binary_data;
    unsigned time_gmt;
    unsigned time_generated;
    long time_bias;
    unsigned record_key;
    unsigned event_id;
    unsigned computer_type;
    unsigned platform_id;
    short version_major;
    short version_minor;
    short event_type;
    short numeric_category;
    unsigned padding000;
};
```

CAUTION: **The binary_data field is present only for compatibility with Windows events. This data cannot be indexed or processed in repository searches. Writing useful data to this field is not recommended.**

event_with_read_extensions

```
struct event_with_read_extensions
{
    struct base_event original_event;
    BSTR formatted_description;
    SAFEARRAY(struct augmented_insertion_string) resolved_strings;
    SAFEARRAY(struct named_string) named_strings;
};
```

named_string

```
struct  named_string
{
    BSTR  name;
    BSTR  value;
};
```

IMPORTANT: In the string names that you define, use only alphanumeric ASCII characters and the underscore (_) character.

insertion_string

```
struct  insertion_string
{
    BSTR value;
    int  index;
    int  padding;
};
```

A regular insertion string.

augmented_insertion_string

```
struct  augmented_insertion_string
{
    int  source_index;
    int  result_index;
    BSTR  value;
};
```

These are normalized parameters that are not originally present in native events. For a description of these parameters, see the Filter Parameters in Repository Viewer topic.

NOTE: The **source_index** field holds the index of the original insertion string. The **result_index** field holds the index of the resulting insertion string after the original has been resolved.

event_with_extensions

```
struct event_with_extensions

{

    struct base_event original_event;

    SAFEARRAY(struct resolved_string) resolved_strings;

};
```

NOTE: The InTrust repository is known to easily handle up to 300 insertion strings per event. Higher numbers of strings have not been tested and cannot be recommended.

As a best practice, make sure that your insertion string mapping is consistent; that is, the same insertion string numbers should have the same meanings across your events. This is beneficial for repository searches.

event_with_extensions2

```
struct event_with_extensions2

{

    struct base_event original_event;

    SAFEARRAY(struct resolved_string) resolved_strings;
```

```
    SAFEARRAY(struct named_string) named_fields;

};
```

resolved_string

```
struct resolved_string
{
    BSTR value;
    int insertion_string_index;
    resolve_type insertion_string_resolve_type;
};

typedef enum
{
    custom = 0,
    parameter,
    ad_object_guid_to_distinguished_name,
    user_sid_to_user_name,
    group_policy_guid_to_group_policy_object_name,
    device_name_to_path
} resolve_type;
```

The **resolve_type** enumeration specifies what kind of resolution is supposed to have taken place to get the resulting **value**. The insertion string resolution mechanism in InTrust is fairly complex, but for the purposes of the repository service API it is enough to follow this example:

```
insertion_string[] insertion_strings =
{
    new insertion_string() { index = 1, padding = 0, value = "original
string" },
    new insertion_string() { index = 2, padding = 0, value = "%%2308" },
// event parameter
    new insertion_string() { index = 3, padding = 0, value = "{9F29FD37-
3CD4-4179-99F1-A6341DCC4EB3}" }, // ad object guid (user guid)
    new insertion_string() { index = 4, padding = 0, value = "S-1-1-0" },
// user sid
    new insertion_string() { index = 5, padding = 0, value = "{29EDB5C5-
B2C1-4001-9C96-EE51A6A7CAC3}" }, // ad object guid (group policy guid)
    new insertion_string() { index = 6, padding = 0, value =
"\\Device\\HarddiskVolume1\\SomeFolder\\SomeFile.txt" } };
resolved_string[] resolved_insertion_strings =
{
    new resolved_string() {insertion_string_index = 2, insertion_string_resolve_
type = resolve_type.parameter, value = "The user has not been granted the
requested logon type at this machine."},
    new resolved_string() {insertion_string_index = 3, insertion_string_resolve_
type = resolve_type.ad_object_guid_to_distinguished_name, value =
"CN=user1,DC=aa,DC=com"},
    new resolved_string() {insertion_string_index = 4, insertion_string_resolve_
type = resolve_type.user_sid_to_user_name, value = "EDM\\User2"},
    new resolved_string() {insertion_string_index = 5, insertion_string_resolve_
type = resolve_type.ad_object_guid_to_distinguished_name, value = "CN={B96B9D14-
```

```
E2A4-47ae-8ACA-CB0460089616},DC=aa,DC=com"},
    new resolved_string() {insertion_string_index = 5, insertion_string_resolve_
type = resolve_type.group_policy_guid_to_group_policy_object_name, value =
"MyGroupPolicyObject"},
    new resolved_string() {insertion_string_index = 6, insertion_string_resolve_
type = resolve_type.ad_object_guid_to_distinguished_name, value =
"D:\\SomeFolder\\SomeFile.txt"},
};
```

Note that in the case of resolving a group policy GUID to a group policy name you need to provide two **resolved_string** instances.

Creating and Removing Repositories

The methods for creating and removing production InTrust repositories (**Add** and **Remove**) are available in the IInTrustRepositoryCollection2 interface, which provides access to all repositories in a particular InTrust organization.

> **CAUTION: For these operations to succeed, the account you are using must be an InTrust organization administrator. To configure this privilege for the account, do one of the following:**
>
> - **In InTrust Deployment Manager, click Manage | Configure Access.**
> - **In InTrust Manager, open the properties of the root node.**

Calling the **Add** method of IInTrustRepositoryCollection2 is not enough to create a repository. After you have made the call and obtained the IInTrustRepository3 interface, you need to complete the configuration of its options and call its **Commit** method. This will complete the creation of the repository.

For details about obtaining a collection of repositories, see Connecting to a Repository.

Instead of a production repository (which is registered with InTrust, managed by an InTrust server and has an entry in the InTrust configuration), you may want to create an idle repository (which has only the raw repository file structure). For that, use the IIdleRepositoryFactory interface, which constructs IIdleRepository interfaces.

Working with Repository Properties

Repositories have very flexible configuration, where some properties are predefined and others can be custom-defined. Access to repository configuration is provided through the IInTrustRepository3 interface, which has getter and setter methods for supported property groupings. The following groupings are available at this time:

- Forwarding properties
  These properties are used by the event forwarding engine in InTrust (see Integration into SIEM Solutions Through Event Forwarding). For details about these properties, see IForwardingSettings.

- Indexing properties
  These properties are the configuration of the repository indexing engine in InTrust (see Repository Indexing for Advanced Search Capabilities). For details about these properties, see IIndexingSettings.
- Custom attributes
  These are arbitrary properties that you can set as necessary for your own purposes. For details, see Using Custom Attributes.

Using Custom Attributes

You can associate custom attributes with InTrust repositories. They are available through the **CustomAttributes** methods of an IInTrustRepository3 interface. They use the IProperty interface and are accessed collectively through IPropertyCollection interfaces.

There are no custom attribute guidelines; what custom attributes you add and how you use them is up to you. However, note that the following limits are set for the generic IProperty interface used by custom attributes:

- Name: 64 characters
- If you set a string of the BSTR type for the value: 1024 characters

It is also recommended that you keep the number of custom attributes low: tens rather than hundreds.

For details about the generic property interfaces used for custom attributes, see IProperty and IPropertyCollection.

Example (C#)

```
/* Connect to repository */
IInTrustEnvironment2 env = new InTrustEnvironment();
IInTrustServer server =   env.ConnectToServerWithCredentials("8.8.8.8", @"domain\user_
name", "password");
IInTrustOrganization org = server.Organization as IInTrustOrganization3;
IInTrustRepository3 rep = org.Repositories2.Item("Default InTrust Audit Repository");

/* Get collection of custom attributes */
IPropertyCollection props = rep.CustomAttributes;

/* Set custom attributes */
props.Set("NumberAttr", 12);
props.Set("StringAttr", "Initial status");
rep.Commit();

/* Get attribute by name */
IProperty stringAttr = props.Item("StringAttr");
/* Get value */
System.Console.WriteLine("String attribute value is {0}", stringAttr.PropertyValue);
/* Set new value */
stringAttr.PropertyValue = "Updated status";
rep.Commit();

/* Enumerate all attributes */
foreach (IProperty prop in props)
{
    System.Console.WriteLine("Attibute : {0}, Value : {1}", prop.PropertyName,
prop.PropertyValue);
}

/* Delete attribute */
props.Remove("NumberAttr");
(props as IADCCommitable).Commit(); /* Commit only the custom properties without the
other repository fields */
```

Log Knowledge Base API

The log knowledge base contains settings for transforming data from original log formats to the repository format. The API does not work with predefined log definitions, which are completely out of its scope; it is designed only for user-defined logs.

To work with the log knowledge base, use the following interfaces:

- IInTrustEventory
- IInTrustEventoryItemCollection
- IInTrustEventoryItem

To begin working with the log knowledge base, get a collection of known organizations (**Organizations** method of the IInTrustEnvironment interface) and pick the necessary one. This involves working with the IInTrustOrganizationCollection interface. Organizations are discovered by an Active Directory query.

The IInTrustOrganization3 that you get has the **Eventory** method, which provides access to the organization-wide log knowledge base.

For details about the format of rules for matching log events and mapping fields, see Log Transformation Rule Format.

> **CAUTION: If you modify the knowledge base for a specific log, this will invalidate all existing index data for that log in all repositories that contain the log. Indexed searches will no longer find this log's events gathered prior to the modification. Data gathered after the modification will be indexed correctly and be searchable.**
>
> **If the unavailability of old data is not a problem for you, you don't have to do anything. Otherwise, you will need to recreate valid indexes for all repositories that contain the log. However, it is not feasible to recreate an index for a large production repository without taking it offline for a long time. If you need to experiment with log knowledge base editing, use a dedicated test organization and small repositories, which can be reindexed quickly.**
>
> **For details about repository reindexing, see Recreating the Index.**

Example

```
static void GetFullEventory()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    string eventory = ev.Eventory;
    Console.WriteLine("Full eventory : " + eventory);
}
static void AddNewLog()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection logs = ev.Logs;
```

```
    IInTrustEventoryItem log = logs.Add("NewLog",
        @"<FieldInfo>
            <Fields>
                <Field FieldName = ""New_field"" DisplayName = ""NewField"" IsIndexed
= ""true""></Field>
            </Fields>
            <EventRules>
                <Event EventID = ""701"">
                    <Field Name = ""Who"" Index = ""11""></Field>
                    <Field Name = ""What"" Index = ""12""></Field>
                    <Field Name = ""Object_Type"" Index = ""13""></Field>
                    <Field Name = ""Object_Name"" Index = ""14""></Field>
                </Event>
            </EventRules>
        </FieldInfo>");
}
static void GetLogAndChangeRule()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection logs = ev.Logs;
    IInTrustEventoryItem log = logs.Item("NewLog");
    log.Rules = @"<FieldInfo>
        <Fields>
            <Field FieldName = ""New_field"" DisplayName = ""NewField"" IsIndexed =
""true""></Field>
        </Fields>
        <EventRules>
            <Event EventID = ""701"">
                <Field Name = ""Who"" Index = ""11""></Field>
            </Event>
        </FieldInfo>";
}
static void EnumLogs()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection logs = ev.Logs;
    foreach (IInTrustEventoryItem cur_log in logs)
    {
        string log_name = cur_log.Name;
        string log_rule = cur_log.Rules;
        Console.WriteLine("Log name : " + log_name);
        Console.WriteLine("Log rule : " + log_rule);
    }
}
static void RemoveLog()
{
    IInTrustEnvironment env = new InTrustEnvironment();
```

```
IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection logs = ev.Logs;
    logs.Remove("NewLog");
}
static void AddNewDataSource()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection dataSources = ev.DataSources;
    IInTrustEventoryItem dataSource = dataSources.Add("{10000000-0000-0000-0000-
000000000001}",@"<FieldInfo>
  <Fields>
    <Field FieldName = ""New_field"" DisplayName = ""NewField"" IsIndexed =
""true""></Field>
  </Fields>
  <EventRules>
    <Event EventID = ""701"">
      <Field Name = ""Who"" Index = ""11""></Field>
      <Field Name = ""What"" Index = ""12""></Field>
      <Field Name = ""Object_Type"" Index = ""13""></Field>
      <Field Name = ""Object_Name"" Index = ""14""></Field>
    </Event>
  </EventRules>
</FieldInfo>");
}
static void GetDataSourceAndChangeRule()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection dataSources = ev.DataSources;
    IInTrustEventoryItem dataSource = dataSources.Item("{10000000-0000-0000-0000-
000000000001}");
    dataSource.Rules = @"<FieldInfo>
        <Fields>
            <Field FieldName = ""New_field"" DisplayName = ""NewField"" IsIndexed =
""true""></Field>
        </Fields>
        <EventRules>
            <Event EventID = ""701"">
                <Field Name = ""Who"" Index = ""11""></Field>
            </Event>
        </FieldInfo>";
}
static void EnumDataSources()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
```

```
IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection dataSources = ev.DataSources;
    foreach (IInTrustEventoryItem curDataSource in dataSources)
    {
        string ds_name = curDataSource.Name;
        string ds_rule = curDataSource.Rules;
        Console.WriteLine("Data source name : " + ds_name);
        Console.WriteLine("Data source rule : " + ds_rule);
    }
}
static void RemoveDataSources()
{
    IInTrustEnvironment env = new InTrustEnvironment();
    IInTrustServer server = env.ConnectToServer("8.8.8.8");
    IInTrustOrganization3 org = server.Organization;
    IInTrustEventory ev = org.Eventory;
    IInTrustEventoryItemCollection dataSources = ev.DataSources;
    dataSources.Remove("{10000000-0000-0000-0000-000000000001}");
}
```

NOTE: In the functions that handle data sources, the data source name must be in GUID format; for example:

```
{10000000-0000-0000-0000-000000000001}
```

Log Transformation Rule Format

Log transformation rules are defined as XML. The structure of a rule is shown in the example below, which contains all of the tags and parameters available.

```
<FieldInfo>
  <Fields>
    <Field FieldName = "TTF" DisplayName = "TTest Field" IsIndexed = "true"></Field>
    <Field FieldName = "TTF2" DisplayName = "TTest Field 2" IsIndexed =
"true"></Field>
  </Fields>
  <EventRules>
    <Event EventID = "701">
      <Field Name = "TTF" Index = "1"></Field>
      <Field Name = "TTF2" Index = "3"></Field>
    </Event>
  </EventRules>
</FieldInfo>
```

Log events are matched by Event ID, and the **Field** tags specify how the original event fields are mapped to repository record fields. The **Index** parameter specifies the index of the target insertion string.

The following is a variation of the example above:

```
<FieldInfo>
    <Fields>
        <Field FieldName = "TTF" DisplayName = "TTest Field" IsIndexed =
```

```
"true"></Field>
        <Field FieldName = "TTF2" DisplayName = "TTest Field 2" IsIndexed =
"true"></Field>
    </Fields>
    <EventRules>
        <Field Name = "TTF" Index = "1"></Field>
        <Field Name = "TTF2" Index = "3"></Field>
    </EventRules>
</FieldInfo>
```

In this second snippet, the rule applies to all event IDs in a log.

Enumerations

CustomizableCredentialsType

```
enum  CustomizableCredentialsType
{
    CurrentCusomizableCredentials,
    DefaultCustomizableCredentials,
    CustomCustomizableCredentials
};
```

DomainEnumerationType

```
enum  DomainEnumerationType
{
    CurrentDomainEnumeration,
    ComputerBrowserDomainEnumeration,
    ComputerListDomainEnumeration
};
```

IndexBuilderType

```
enum  IndexBuilderType
{
    CurrentIndexBuilder,
    ServerIndexBuilder,
    SiteIndexBuilder
};
```

IndexLocationType

```
enum  IndexLocationType
{
    CurrentIndexLocation,
    RepositoryIndexLocation,
    CustomIndexLocation
};
```

RepositoryCommitType

```
enum  RepositoryCommitType
{
    ToRepositoryRepositoryCommitType,
    ToServerRepositoryCommitType
};
```

ScriptLanguage

```
enum  ScriptLanguage
{
    ECMAScript,
    JScript,
    PSScript,
    VBScript
};
```

SiteCollectionType

```
enum  SiteCollectionType
{
    VisibleSites,
    AllSites
};
```

SiteObjectType

```
enum  SiteObjectType
{
    UnknownSiteObject,
    MsnWholeNetworkSiteObject,
    DomainSiteObject,
    ComputerSiteObject,
    IPAddressRangeSiteObject,
    ComputerListFileSiteObject,
    OrganizationalUnitSiteObject,
    ActiveDirectorySiteSiteObject,
    EnumerationScriptSiteObject,
    DomainControllersInDomainSiteObject,
    DomainControllersInActiveDirectorySiteSiteObject
};
```

SiteType

```
SiteType
{
    MicrosoftNetworkSite,
    UnixNetworkSite
};
```

Interfaces

The following is a list of all interfaces available with the InTrust repository API:

Interface	Details
IActiveDirectorySiteSiteObject	Represents an Active Directory site from which to put computers in an InTrust site.
IBulkEventWithReadExtensions	Results of a repository search as an array of event_with_read_extensions structures.
IBulkRecord	Records packed into a single batch as an array of record structures for writing to the repository.
IBulkRecord2	Results of a repository search as an array of record2 structures.
IComputerListDomainEnumeration	Represents the configuration of site membership enumeration through a computer list.
IComputerListFileSiteObject	Represents a file that contains a list of computers to include in an InTrust site.
IComputerSiteObject	Represents a single computer in an InTrust site.
ICookie	Acts as the owner of a repository search and can stop the search.
ICredentials	Represents a credential set used by InTrust for access to resources.
ICustomCredentials	Wrapper for a credential set used by InTrust for access to resources.
ICustomIndexLocation	Gets or sets the location of the index of a repository.
ICustomizableCredentials	This is a wrapper for a credential set used by InTrust

Interface	Details
	for access to resources.
IDomainEnumeration	Indicates which method of domain enumeration is used for populating an InTrust site.
IDomainSiteObject	Represents computers in an InTrust site that are indicated by an Active Directory domain.
IEnumerationScriptSiteObject	Represents a script that enumerates the computers to include in an InTrust site.
IEventToRecordFormatter	Transforms event records to a representation suitable for insertion into a repository by the **PutRecords2** method of IRepositoryRecordInserter.
IForwardingFilterCollection	Provides a collection of all search-based filters associated with an InTrust repository.
IForwardingSettings	Provides access to the event forwarding settings for a repository.
IIdleRepository	An *idle* repository has the correct structure on the file system, but is not registered with an InTrust organization. Currently, you can search in idle repositories using the repository API, but you cannot write to them.
IIdleRepositoryFactory	Creates an idle InTrust repository.
IIndexBuilder	Represents the index-building configuration for the repository. Indexing can be performed by an InTrust server or delegated to InTrust agents in a specific site.
IIndexingSettings	Provides access to the indexing configuration of a repository.

Interface	Details
IIndexManager	Provides access to indexing-related operations.
IIndexManagerFactory	Creates an instance of IIndexManager for a production or idle repository.
IInTrustEnvironment	See IInTrustEnvironment3.
IInTrustEnvironment3	Entry point for access to InTrust organizations, servers and repositories.
IInTrustEventory	Provides access to the log knowledge base, which contains rules that govern the transformation of log entries into repository and event records.
IInTrustEventoryItem	Represents an entry in the log knowledge base.
IInTrustEventoryItemCollection	Provides a collection of IInTrustEventoryItem interfaces.
IInTrustOrganization	Legacy interface supplanted by IInTrustOrganization3.
IInTrustOrganization3	Provides access to an InTrust organization.
IInTrustOrganizationCollection	Provides a collection of all available InTrust organizations.
IInTrustRepository3	Provides the searching and writing capabilities of a repository.
IInTrustRepositoryCollection2	Provides a collection of all repositories available in the InTrust organization.
IInTrustRepositorySearcher	Provides repository search capabilities.
IInTrustScriptCollection	Provides a collection of scripts used in InTrust operations.
IInTrustServer	Provides access to an InTrust server.

Interface	Details
IInTrustServerCollection	Provides a collection of all InTrust servers in the InTrust organization.
IInTrustSiteCollection	Represents the sites in an InTrust organization.
IIPAddressRangeSiteObject	Represents a range of IP addresses that are included in an InTrust site.
IJob2	Represents a subset of the configuration of an InTrust job.
ITransportInfoCollection	Provides a collection of all transport types supported by InTrust event forwarding.
IMessageFormatCustomInfo	Represents a customizable script-based message formatter used for event forwarding.
IMessageFormatInfo	Provides access to the formatting configuration for forwarded events.
IMessageFormatTypeInfo	Defines a message format for forwarded events and can be used as a template for creating new formats.
IMessageFormatTypeInfoCollection	Provides a collection of all message format types supported by InTrust event forwarding.
IMicrosoftNetworkSite	Represents an InTrust site of the Microsoft Windows Network type.
IMultiRepositorySearcher	A container for search objects that lets you search in all of the specified repositories simultaneously.
IMultiRepositorySearcherFactory	Creates an instance of IMultiRepositorySearcher.
IObservable	Defines a provider for push-based notification.

Interface	Details
IObserver	Provides a mechanism for receiving push-based notifications. You need to create your own implementation of this interface.
IOrganizationalUnitSiteObject	Represents an organizational unit from which to put computers in an InTrust site.
IProperty	Property attached to an InTrust repository. A property is a way to tag repositories for arbitrary purposes.
IPropertyCollection	Collection of properties associated with an InTrust repository. Access to the collections is gained through specialized methods of the IInTrustRepository3 interface (such as **CustomAttributes**).
IRepositoryRecordInserter	Provides write access to the repository that it is associated with and manages one or more IRepositoryRecordInserterLight interfaces, which do the actual writing.
IRepositoryRecordInserter2	Provides write access to the repository that it is associated with and manages one or more IRepositoryRecordInserterLight interfaces, which do the actual writing. This method can submit records to a queue on the server or put them directly in the repository.
IRepositoryRecordInserterLight	Generates valid record structures from predefined and significant values and writes them to the repository.
IScript	Represents a script used in InTrust operations.
IScriptArgument	Represents an argument used

Interface	Details
	with an InTrust site enumeration script.
IScriptArgumentCollection	Represents the arguments defined for an InTrust script.
IScriptParameter	Represents a customizable parameter defined for an InTrust script.
IScriptParameterCollection	Represents the parameters defined for an InTrust script.
ISite	Represents an InTrust site, which can be a regular site visible in InTrust Manager or a hidden internal site associated with a collection visible in InTrust Deployment Manager.
ISiteComputer	Represents a computer that is included in an InTrust site.
ISiteComputerCollection	Represents an InTrust site associated with a collection visible in InTrust Deployment Manager.
ISiteIndexBuilder	Represents the distributed indexing configuration for a repository.
ISiteObject	Represents a computer-specifying object that can be included in a site.
ISiteObjectCollection	Represents the computer-specifying objects included in a site.
ITask2	Represents a subset of the configuration of an InTrust scheduled task.
IIndexBuilderAccess	Represents the security settings for performing repository indexing.
ITransportInfo	Represents a transport type supported by InTrust event forwarding.

Interface	Details
ITransportInfoCollection	Provides a collection of all transport types supported by InTrust event forwarding.

IActiveDirectorySiteSiteObject

Represents an Active Directory site from which to put computers in an InTrust site.

Methods

Domain (getter)

Returns the domain of the site.

Syntax

```
HRESULT  Domain(
     [out,  retval]BSTR*  bstrDomain
);
```

Parameter

Name	Type	Meaning
bstrDomain	BSTR*	Domain of the site.

Domain (setter)

Sets the domain of the site.

Syntax

```
HRESULT  Domain(
     [in]BSTR  bstrDomain
);
```

Parameter

Name	Type	Meaning
bstrDomain	BSTR	Domain of the site.

Site (getter)

Returns the name of the site.

Syntax

```
HRESULT  Site(
     [out,  retval]BSTR*  bstrSite
);
```

Parameter

Name	Type	Meaning
bstrSite	BSTR*	Name of the site.

Site (setter)

Sets the name of the site.

Syntax

```
HRESULT  Site(
     [in]BSTR  bstrSite
);
```

Parameter

Name	Type	Meaning
bstrSite	BSTR	Name of the site.

IBulkEventWithReadExtensions

Use this interface to represent the results of a repository search as an array of event_with_read_extensions structures.

Method

GetRecords

Gets records represented by event_with_read_extensions structures.

Syntax

```
GetRecords(
     [out,retval]  SAFEARRAY(struct  event_with_read_extensions)*  events
);
```

Parameter

Name	Type	Meaning
events	SAFEARRAY(struct event_with_read_extensions)*	Records represented by event_with_read_extensions structures.

IBulkRecord

Represents a batch of repository records as an array of record structures for writing.

Method

GetRecords

Gets records that match search terms.

Syntax

```
GetRecords(
    [out,retval] SAFEARRAY(struct record)* records
);
```

Parameter

Name	Type	Meaning
events	SAFEARRAY(struct record)*	Packed repository records.

IBulkRecord2

Represents the results of a repository search as an array of record2 structures.

Method

GetRecords

Gets records that match search terms.

Syntax

```
GetRecords(
    [out,retval] SAFEARRAY(struct record2)* records
);
```

Parameter

Name	Type	Meaning
events	SAFEARRAY(struct record2)*	Discovered repository records.

IComputerListDomainEnumeration

Represents the configuration of site membership enumeration through a computer list.

Methods

IgnorePasswordsOlderThanInterval (getter)

Returns the maximum age (in days) for the passwords of the accounts of the computers in the site. If a computer account's password is older than that, the computer is excluded from the site.

Syntax

```
HRESULT  IgnorePasswordsOlderThanInterval(
    [out,  retval]  long*  ignoreInterval
);
```

Parameter

Name	Type	Meaning
ignoreInterval	long*	Maximum password age in days.

IgnorePasswordsOlderThanInterval (setter)

Sets the maximum age (in days) for the passwords of the accounts of the computers in the site. If a computer account's password is older than that, the computer is excluded from the site.

Syntax

```
HRESULT  IgnorePasswordsOlderThanInterval(
    [in]  long  ignoreInterval
);
```

Parameter

Name	Type	Meaning
ignoreInterval	long	Maximum password age in days.

IsIgnoringByOldPasswordsEnabled (getter)

Returns whether filtering by password age is enabled. If a computer account's password is older than a specified interval, the computer is excluded from the site.

Syntax

```
IsIgnoringByOldPasswordsEnabled(
    [out, retval]VARIANT_BOOL* bIgnoringEnabled
);
```

Parameter

Name	Type	Meaning
bIgnoringEnabled	VARIANT_BOOL*	Whether filtering by password age is enabled.

IsIgnoringByOldPasswordsEnabled (setter)

Sets whether filtering by password age is enabled. If a computer account's password is older than a specified interval, the computer is excluded from the site.

Syntax

```
HRESULT IsIgnoringByOldPasswordsEnabled(
    [in] VARIANT_BOOL bIgnoringEnabled
);
```

Parameter

Name	Type	Meaning
bIgnoringEnabled	VARIANT_BOOL	Whether filtering by password age is enabled.

IComputerListFileSiteObject

Represents a file that contains a list of computers to include in an InTrust site.

Methods

Path (getter)

Returns the path to the computer list file.

Syntax

```
HRESULT Path(
    [out, retval]BSTR* bstrPath
);
```

Parameter

Name	Type	Meaning
bstrPath	BSTR*	Path to the computer list file.

Path (setter)

Sets the path to the computer list file.

Syntax

```
HRESULT  Path(
     [in]BSTR  bstrPath
);
```

Parameter

Name	Type	Meaning
bstrPath	BSTR	Path to the computer list file.

IComputerSiteObject

Represents a single computer in an InTrust site.

Methods

Computer (getter)

Gets the computer wrapped in this site object.

Syntax

```
HRESULT  Computer(
     [out,  retval]BSTR*  bstrComputer
);
```

Parameter

Name	Type	Meaning
bstrComputer	BSTR*	Computer in the site object.

Computer (setter)

Sets the computer wrapped in this site object.

Syntax

```
HRESULT  Computer(
     [in]BSTR  bstrComputer
);
```

Parameter

Name	Type	Meaning
bstrComputer	BSTR	Computer in the site object.

ICookie

Acts as the owner of a repository search and can stop the search.

Method

Stop

Stops the search that this interface is associated with. This method is called automatically when the last reference to the interface is destroyed.

> **CAUTION: This is a synchronous method. It must never be called from notifications received through IObserver, because this will result in a deadlock.**

Syntax

```
HRESULT Stop()
```

ICredentials

Represents a credential set used by InTrust for access to resources.

Methods

AccountName (getter)

Returns the account name.

Syntax

```
HRESULT AccountName(
    [out, retval] BSTR* accountName
);
```

Parameters

Name	Type	Meaning
accountName	BSTR*	Account name.

AccountName (setter)

Sets the account name.

Syntax

```
HRESULT AccountName(
    [in] BSTR accountName
);
```

Parameters

Name	Type	Meaning
accountName	BSTR	Account name.

AccountPassword

Sets the password for the account.

Syntax

```
HRESULT AccountPassword(
    [in] BSTR accountPassword
);
```

Parameters

Name	Type	Meaning
accountPassword	BSTR	Password for the account.

ICustomIndexLocation

Gets or sets the location of the index of a repository.

Methods

IndexPath (getter)

Returns the path to the index.

Syntax

```
HRESULT  IndexPath(
     [out,  retval]  BSTR*  indexPath
);
```

Parameter

Name	Type	Meaning
indexPath	BSTR*	Path to the index.

IndexPath (setter)

Returns the path to the index.

Syntax

```
HRESULT  IndexPath(
     [in]  BSTR  indexPath
);
```

Parameter

Name	Type	Meaning
indexPath	BSTR	Path to the index.

ICustomCredentials

This is a wrapper for a credential set used by InTrust for access to resources.

Method

Credentials

Provides access to the credential set.

Syntax

```
HRESULT Credentials(
    [out, retval] ICredentials** ppCredentials
);
```

Parameter

Name	Type	Meaning
ppCredentials	ICredentials**	Credential set.

ICustomizableCredentials

Indicates what type of credential set is used. If the result is **CustomCustomizableCredentials**, you can cast this to ICustomCredentials to work with the credential set.

Method

Type

Gets the type of credential set used.

Syntax

```
HRESULT Type(
    [out, retval] enum CustomizableCredentialsType* pType
);
```

Parameter

Name	Type	Meaning
pType	enum CustomizableCredentialsType*	Type of credential set used.

IDomainEnumeration

Indicates which method of domain enumeration is used for populating an InTrust site. If the result is **ComputerListDomainEnumeration**, you can cast this to IComputerListDomainEnumeration to obtain and modify the configuration.

Method

Type

Gets the method of domain enumeration used for populating an InTrust site.

Syntax

```
HRESULT  Type(
     [out,  retval]  enum  DomainEnumerationType*  pType
);
```

Parameter

Name	Type	Meaning
pType	enum DomainEnumerationType*	Which method of domain enumeration is used.

IDomainSiteObject

Represents computers in an InTrust site that are indicated by an Active Directory domain.

Methods

Domain (getter)

Gets the domain that indicates the computers.

Syntax

```
HRESULT  Domain(
     [out,  retval]BSTR*  bstrDomain
);
```

Parameter

Name	Type	Meaning
bstrDomain	BSTR*	Domain that indicates the computers.

Domain (setter)

Sets the domain that indicates the computers.

Syntax

```
RESULT  Domain(
     [in]BSTR  bstrDomain
);
```

Parameter

Name	Type	Meaning
bstrDomain	BSTR	Domain that indicates the computers.

IEnumerationScriptSiteObject

Represents a script that enumerates the computers to include in an InTrust site.

Methods

Script (getter)

Returns the script that enumerates site computers.

Syntax

```
HRESULT  Script(
     [out,  retval]IScript**  ppScript
);
```

Parameter

Name	Type	Meaning
ppScript	IScript**	Script that enumerates site computers.

Script (setter)

Sets the script that enumerates site computers.

Syntax

```
HRESULT  Script(
     [in]IScript*  pScript
);
```

Parameter

Name	Type	Meaning
pScript	IScript*	Script that enumerates site computers.

Name (getter)

Returns the name of the script.

Syntax

```
HRESULT  Name(
     [out,  retval]BSTR*  bstrName
);
```

Parameter

Name	Type	Meaning
bstrName	BSTR*	Name of the script.

Name (setter)

Sets the name of the script.

Syntax

```
HRESULT  Script(
     [in]BSTR  bstrName
);
```

Parameter

Name	Type	Meaning
bstrName	BSTR	Name of the script.

Arguments

Provides access to the arguments used with the script.

Syntax

```
HRESULT  Arguments(
     [out,  retval]IScriptArgumentCollection**  ppArguments
);
```

Parameter

Name	Type	Meaning
ppArguments	IScriptArgumentCollection**	Arguments used with the script.

IEventToRecordFormatter

Transforms event records to a representation suitable for insertion into a repository by the **PutRecords2** method of IRepositoryRecordInserter. For details about event records, see Event Record Data Structures.

Method

Format

Performs the event-to-record transformation.

Syntax

```
HRESULT  Format(
     [in]  SAFEARRAY(struct  event_with_extensions)  events,
     [out,  retval]  IBulkRecord**  ppBulkRecord
);
```

Parameters

Name	Type	Meaning
events	SAFEARRAY(struct event_with_extensions)	Event records to put into the repository.
ppBulkRecord	IBulkRecord**	Repository records prepared for insertion.

IForwardingFilterCollection

Provides a collection of all search-based filters associated with an InTrust repository.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT  _NewEnum(
     [out,  retval]  LPUNKNOWN*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Add

Adds a new filter based on a Repository Viewer search.

Syntax

```
HRESULT  Add(
     [in]  BSTR  bstrPath
);
```

Parameter

Name	Type	Meaning
bstrPath	BSTR	Path to the Repository Viewer search to make the new filter from. At this time, an internal representation of the path is expected, and there is no usable way to look it up for a specific search. Later implementations should fix this.

Remove

Removes the specified filter from the collection. To specify the filter, pass in the path to its corresponding Repository Viewer search.

Syntax

```
HRESULT  Remove(
     [in]  BSTR  bstrPath
);
```

Parameter

Name	Type	Meaning
bstrPath	BSTR	Path to the Repository Viewer search to make the new filter from.

IForwardingSettings

Provides access to the event forwarding settings for a repository.

Methods

ForwardingEnabled (getter)

Returns whether event forwarding is enabled for the repository.

Syntax

```
HRESULT ForwardingEnabled(
    [out, retval] VARIANT_BOOL* pForwardingEnabled
);
```

Parameter

Name	Type	Meaning
pForwardingEnabled	VARIANT_BOOL*	Whether event forwarding is enabled for the repository.

ForwardingEnabled (setter)

Enables or disables event forwarding for the repository.

Syntax

```
HRESULT ForwardingEnabled(
    [in] VARIANT_BOOL ForwardingEnabled
);
```

Parameter

Name	Type	Meaning
pForwardingEnabled	VARIANT_BOOL	Whether to enable or disable event forwarding for the repository.

ForwardingFilters

Returns the collection of repository searches used as forwarding filters. Management of these filters is outside the scope of the InTrust SDK.

Syntax

```
HRESULT ForwardingFilters(
    [out, retval] IForwardingFilterCollection** ppForwardingFilters
);
```

Parameter

Name	Type	Meaning
ppForwardingFilters	IForwardingFilterCollection**	Repository searches used as forwarding filters.

ForwardingServer (getter)

Returns the InTrust server that manages event forwarding from the repository.

Syntax

```
HRESULT  ForwardingServer(
     [out,  retval]  IInTrustServer3**  ppForwardingServer
);
```

Parameters

Name	Type	Meaning
ppForwardingServer	IInTrustServer3**	InTrust server that manages event forwarding from the repository.

ForwardingServer (setter)

Sets the InTrust server that manages event forwarding from the repository.

Syntax

```
HRESULT  ForwardingServer(
     [in]  IInTrustServer3*  pForwardingServer
);
```

Parameters

Name	Type	Meaning
pForwardingServer	IInTrustServer3*	InTrust server that manages event forwarding from the repository.

Host (getter)

Returns the name or IP address of the destination host for event forwarding.

Syntax

```
 HRESULT  Host(
     [out,  retval]  BSTR*  bstrHost
);
```

Parameter

Name	Type	Meaning
bstrHost	BSTR*	Name or IP address of the destination host for event forwarding.

Host (setter)

Sets the name or IP address of the destination host for event forwarding.

Syntax

```
HRESULT Host(
    [in] BSTR bstrHost
);
```

Parameter

Name	Type	Meaning
bstrHost	BSTR	Name or IP address of the destination host for event forwarding.

MessageFormat (getter)

Returns the details of the message format used for event forwarding from the repository.

Syntax

```
HRESULT MessageFormat(
    [out, retval] IMessageFormatInfo** ppMessageFormatInfo
);
```

Parameter

Name	Type	Meaning
ppMessageFormatInfo	IMessageFormatInfo **	Details of the message format used for event forwarding from the repository.

MessageFormat (setter)

Sets the message format to use for event forwarding from the repository.

Syntax

```
HRESULT MessageFormat(
    [in] IMessageFormatInfo* pMessageFormatInfo
);
```

Parameter

Name	Type	Meaning
pMessageFormatInfo	IMessageFormatInfo *	Details of the message format used for event forwarding from the repository.

Port (getter)

Returns the destination port for event forwarding.

Syntax

```
HRESULT  Port(
    [out,  retval]  BSTR*  bstrPort
);
```

Parameter

Name	Type	Meaning
bstrPort	BSTR*	Destination port for event forwarding.

Port (setter)

Sets the destination port for event forwarding.

Syntax

```
HRESULT  Port(
    [in]  BSTR  bstrPort
);
```

Parameter

Name	Type	Meaning
bstrPort	BSTR	Destination port for event forwarding.

TransportInfo (getter)

Returns the details of the transport selected for event forwarding from the repository.

Syntax

```
HRESULT  Transport(
    [out,  retval]  ITransportInfo**  ppTransportInfo
);
```

Parameter

Name	Type	Meaning
ppTransportInfo	ITransportInfo**	Transport selected for event forwarding from the repository.

TransportInfo (setter)

Sets the transport for event forwarding from the repository.

Syntax

```
HRESULT  Transport(
     [in]  ITransportInfo*  pTransportInfo
);
```

Parameter

Name	Type	Meaning
pTransportInfo	ITransportInfo*	Transport selected for event forwarding from the repository.

IIdleRepository

Represents an idle repository. An *idle* repository has the correct structure on the file system, but is not registered with an InTrust organization. Currently, you can search in idle repositories using the repository API, but you cannot write to them.

Method

Searcher

Returns a searcher interface for the idle repository.

Syntax

```
HRESULT Searcher(
    [in, optional] VARIANT pIndexManager,
    [out, retval] IInTrustRepositorySearcher** ppSearcher);
```

Parameters

Name	Type	Meaning
pIndexManager	VARIANT	Interface that contains details about the index to use for searching in the repository. See IIndexManager for details.
ppSearcher	IInTrustRepositorySearcher	Searcher interface that you can supply your query to.

IIdleRepositoryFactory

Creates an idle InTrust repository. An *idle* repository has the correct structure on the file system, but is not registered with an InTrust organization.

Method

MakeIdleRepository

Returns an idle InTrust repository.

Syntax

```
HRESULT MakeIdleRepository(
    [in] BSTR bstrPath,
    [in] BSTR bstrUser,
    [in] BSTR bstrPassword,
    [out, retval] IIdleRepository **ppRepository);
```

Parameters

Name	Type	Meaning
bstrPath	BSTR	Search query.
bstrUser	BSTR	User account to use for the operation.
bstrPassword	BSTR	Password to use for the operation.
ppRepository	IIdleRepository	The newly-created idle repository.

IIndexBuilder

Represents the index-building configuration for the repository. Indexing can be performed by an InTrust server or delegated to InTrust agents in a specific site. Offloading indexing to additional agents can help with InTrust server load balancing.

All this interface does is say what is used for building the index for the repository. If the type of builder is **SiteIndexBuilder**, then you can configure it by casting **IIndexBuilder** to ISiteIndexBuilder and working with its methods.

Method

Type

Syntax

```
HRESULT  Type(
    [out,  retval]  enum  IndexBuilderType*  pType
);
```

Parameter

Name	Type	Meaning
pType	enum IndexBuilderType*	What resources are used for index building.

IIndexingSettings

Provides access to the indexing configuration of a repository.

Methods

IndexingServer (getter)

Returns the indexing server for the repository.

Syntax

```
HRESULT  IndexingServer(
     [out,  retval]  IInTrustServer3**  ppIndexingServer
);
```

Parameter

Name	Type	Meaning
ppIndexingServer	IInTrustServer3**	Indexing server for the repository.

IndexingServer (setter)

Sets the specified indexing server for the repository.

Syntax

```
HRESULT  IndexingServer(
     [in]  IInTrustServer3*  pIndexingServer
);
```

Parameter

Name	Type	Meaning
pIndexingServer	IInTrustServer3*	Indexing server to set for the repository.

IndexBuilder

Provides access to the index-building configuration for the repository. For details, see IIndexBuilder.

Syntax

```
HRESULT IndexBuilder(
    [in, defaultvalue(CurrentIndexBuilder)] enum IndexBuilderType,
    [out, retval] IIndexBuilder** pIndexPathType
);
```

Parameter

Name	Type	Meaning
	enum IndexBuilderType	Selects the method's operation mode: • Return the currently set index-building configuration • Switch to InTrust server indexing • Switch to delegated indexing by agents in an InTrust site
pIndexPathType	IIndexBuilder**	Index-building configuration for the repository.

IndexLocation

Returns the location of the index for the repository.

Syntax

```
HRESULT IndexLocation(
    [in, defaultvalue(CurrentIndexLocation)] enum IndexLocationType,
    [out, retval] IIndexLocation** pIndexPathType
);
```

Parameter

Name	Type	Meaning
CurrentIndexLocation	enum IndexLocationType	
pIndexPathType	IIndexLocation**	Indexing server for the repository.

IsIndexingEnabled (getter)

Indicates whether indexing is enabled for the repository.

Syntax

```
HRESULT IsIndexingEnabled(
    [out, retval] VARIANT_BOOL* bEnabled
);
```

Parameter

Name	Type	Meaning
bEnabled	VARIANT_BOOL*	Whether indexing is enabled for the repository.

IsIndexingEnabled (setter)

Enables or disables indexing for the repository.

Syntax

```
HRESULT  IsIndexingEnabled(
     [in]  VARIANT_BOOL  bEnabled
);
```

Parameter

Name	Type	Meaning
bEnabled	VARIANT_BOOL	Whether to enable or disable indexing for the repository.

IIndexLocation

Represents the location of the index, which can be a folder in the repository share, as by default, or a custom network share.

All this interface does is say whether the index is in the default or a custom location. If it is custom, cast the **IIndexLocation** instance to ICustomIndexLocation to get or set the path.

Method

Type

Syntax

```
HRESULT  Type(
    [out,  retval]  enum  IndexLocationType*  pType)
);
```

Parameter

Name	Type	Meaning
pType	enum IndexLocationType*	Where the index is located: default folder or share.

IIndexManager

Provides access to indexing-related operations.

Methods

GetID

Returns the ID of the index manager.

Syntax

```
HRESULT GetID(
    [out, retval] BSTR*
);
```

Parameter

Name	Type	Meaning
	BSTR*	ID of the index manager.

Shutdown

Shuts down the index manager.

Syntax

```
HRESULT Shutdown();
```

IIndexManagerFactory

Creates an instance of IIndexManager for a production or idle repository.

Methods

GetRemoteIndexManager

Creates an IIndexManager instance for a production repository.

Syntax

```
HRESULT GetRemoteIndexManager(
    [in] BSTR pszServerName,
    [in] BSTR pszRepositoryIdentity,
    [out] IIndexManager** ppManager
);
```

Parameters

Name	Type	Meaning
pszServerName	BSTR	Name of an InTrust server in the same organization as the repository.
pszRepositoryIdentity	BSTR	ID of the production repository that you need.
ppManager	IIndexManager **	Index manager for the production repository.

GetLocalIndexManager

Creates an IIndexManager instance for an idle repository.

Syntax

```
HRESULT  GetLocalIndexManager(
     [in]  BSTR  pszIndexPath,
     [in]  BSTR  pszRepositoryPath,
     [in]  BSTR  pszAcount,
     [in]  BSTR  pszPassword,
     [in]  enum  modeOpen  mode,
     [out,  retval]  IIndexManager**
);
```

Parameters

Name	Type	Meaning
pszIndexPath	BSTR	Path to the index data for the idle repository.
pszRepositoryPath	BSTR	Path to the idle repository file structure.
pszAcount	BSTR	User name for access to the idle repository.
pszPassword	BSTR	Password for access to the idle repository.
mode	enum	MODE_OPEN = 0 MODE_CREATE = 1
	IIndexManager**	Index manager to use for indexing-related operations.

IInTrustEnvironment

Entry point for access to InTrust organizations, servers and repositories. Whenever you obtain this legacy interface, you should cast it to IInTrustEnvironment3 and use that instead.

Methods

ConnectToServer

Provides access to the specified InTrust server.

> **CAUTION: For this operation to succeed, the account you are using must be a member of the AMS Readers local group on the InTrust server you want to connect to.**
>
> **Alternatively, it can be an InTrust organization administrator. To configure this privilege for the account, do one of the following:**
>
> - **In InTrust Deployment Manager, click Manage | Configure Access.**
> - **In InTrust Manager, open the properties of the root node.**

Syntax

```
HRESULT ConnectToServer(
    [in] BSTR bstrServerBinding,
    [out, retval] IInTrustServer** ppServer
);
```

Parameters

Name	Type	Meaning
bstrServerBinding	BSTR	Name of the InTrust server.
	IInTrustServer**	InTrust server interface.

Organizations

Provides a collection of available InTrust organizations.

Syntax

```
HRESULT Organizations(
    [out, retval] IInTrustOrganizationCollection** ppOrganization
);
```

Parameter

Name	Type	Meaning
ppOrganization	IInTrustOrganizationCollection**	Collection of available InTrust organizations.

Eventory

Provides access to the log knowledge database associated with the InTrust organization.

Syntax

```
HRESULT Eventory(
    [out, retval] IInTrustEventory **ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustEventory**	Log knowledge database associated with the InTrust organization.

IInTrustEnvironment3

Entry point for access to InTrust organizations, servers and repositories.

Methods

ConnectToServer

Provides access to the specified InTrust server. The credentials of the current user are used for the operation.

> **CAUTION: For this operation to succeed, the account you are using must be a member of the AMS Readers local group on the InTrust server you want to connect to.**
>
> **Alternatively, it can be an InTrust organization administrator. To configure this privilege for the account, do one of the following:**
>
> - **In InTrust Deployment Manager, click Manage | Configure Access.**
> - **In InTrust Manager, open the properties of the root node.**

Syntax

```
HRESULT ConnectToServer(
    [in] BSTR bstrServerBinding,
    [out, retval] IInTrustServer** ppServer
);
```

Parameters

Name	Type	Meaning
bstrServerBinding	BSTR	Name of the InTrust server.
	IInTrustServer**	InTrust server interface.

ConnectToServerLocal

Provides access to the InTrust server running locally.

Syntax

```
ConnectToServerLocal(
    [out, retval] IInTrustServer** ppServer
);
```

Parameters

Name	Type	Meaning
ppServer	IInTrustServer**	Interface for the local InTrust server.

ConnectToServerWithCredentials

Provides access to the specified InTrust server using the specified credentials.

Syntax

```
HRESULT ConnectToServerWithCredentials(
    [in] BSTR bstrServerBinding,
    [in] BSTR bstrUserName,
    [in] BSTR bstrUserPasword,
    [out, retval] IInTrustServer** ppServer);
```

Parameters

Name	Type	Meaning
bstrServerBinding	BSTR	Name of the InTrust server.
bstrUserName	BSTR	User name of the account to use for the operation.
bstrUserPasword	BSTR	Password of the account to use for the operation.
ppServer	IInTrustServer**	Interface for the InTrust server.

Organizations

Provides a collection of available InTrust organizations.

Syntax

```
HRESULT Organizations(
    [out, retval] IInTrustOrganizationCollection** ppOrganization
);
```

Parameter

Name	Type	Meaning
ppOrganization	IInTrustOrganizationCollection**	Collection of available InTrust organizations.

IInTrustEventory

Provides access to the log knowledge base, which contains rules that govern the transformation of log entries into repository and event records.

Methods

Eventory

Returns a string representation of the log knowledge base.

Syntax

```
HRESULT  Eventory(
     [out,  retval]  BSTR*  bstrEventory
);
```

Parameters

Name	Type	Meaning
bstrEventory	BSTR*	String representation of the log knowledge base.

Logs

Provides access to the log knowledge base entries through an IInTrustEventoryItemCollection.

Syntax

```
HRESULT  Logs(
     [out,  retval]  IInTrustEventoryItemCollection**  pVal
);
```

Parameters

Name	Type	Meaning
pVal	IInTrustEventoryItemCollection**	Log knowledge base entries.

DataSources

```
HRESULT  DataSources(
     [out,  retval]  IInTrustEventoryItemCollection**  pVal
);
```

Parameters

Name	Type	Meaning
pVal	IInTrustEventoryItemCollection**	Log knowledge base entries.

IInTrustEventoryItem

Represents an entry in the log knowledge base.

Methods

Name

Returns the name of the log knowledge database entry.

Syntax

```
HRESULT Name(
    [out, retval] BSTR* bstrName

);
```

Parameter

Name	Type	Meaning
bstrName	BSTR*	Name of the log knowledge database entry.

Rules (out parameter)

Returns the rules defined for the log knowledge database entry. For details about the rule format, see Log Transformation Rule Format.

Syntax

```
HRESULT Rules(
    [out, retval] BSTR* bstrRules

);
```

Parameter

Name	Type	Meaning
bstrRules	BSTR*	Textual representation of the rules defined for the log knowledge database entry.

Rules (in parameter)

Sets the rules defined for the log knowledge database entry. For details about the rule format, see Log Transformation Rule Format.

Syntax

```
HRESULT Rules(
    [in] BSTR bstrRules

);
```

Parameter

Name	Type	Meaning
bstrRules	BSTR	Textual representation of the rules defined for the log knowledge database entry.

IInTrustEventoryItemCollection

Provides a collection of IInTrustEventoryItem interfaces.

Methods

Item

Gets a log knowledge base entry from the collection by name.

Syntax

```
HRESULT  Item(
     [in]  BSTR  bstrLogName,
     [out,  retval]  IInTrustEventoryItem**  ppEventoryItem
);
```

Parameters

Name	Type	Meaning
bstrLogName	BSTR	Name of the log knowledge base entry. This name must exist in the collection.
ppEventoryItem	IInTrustEventoryItem **	The log knowledge base entry.

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT _NewEnum(
    [out, retval] LPUNKNOWN* pVal
);
```

Parameters

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Add

Adds the specified entry.

Syntax

```
HRESULT Add(
    [in] BSTR bstrItemName,
    [in] BSTR bstrItemRules,
    [out, retval] IInTrustEventoryItem** ppEventoryItem
);
```

Parameters

Name	Type	Meaning
bstrItemName	BSTR	Name of the entry to add. The name must be unique.
bstrItemRules	BSTR	Textual definition of the entry.
ppEventoryItem	IInTrustEventoryItem	The new log knowledge base entry.

Remove

Removes an entry from the collection by name.

Syntax

```
HRESULT Remove(
    [in] BSTR bstrItemName
);
```

Parameter

Name	Type	Meaning
bstrItemName	BSTR	Name of the entry to remove.

IInTrustOrganization

Provides access to an InTrust organization.

> **IMPORTANT:** This interface is deprecated. Before you start working with an organization, cast **IInTrustOrganization** to IInTrustOrganization3.

Methods

Name

Returns the name of the InTrust organization.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR	Name of the InTrust organization.

Servers

Provides access to a collection of the InTrust servers in an InTrust organization.

Syntax

```
HRESULT  Servers(
     [out,  retval]  IInTrustServerCollection**  ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustServerCollection**	Collection of InTrust servers.

Repositories

Provides access to a collection of repositories in an InTrust organization.

Syntax

```
HRESULT  Repositories(
     [out,  retval]  IInTrustRepositoryCollection**  ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustRepositoryCollection**	Collection of repositories.

Eventory

Provides access to the organization-wide log knowledge base. See Log Knowledge Base API for details.

Syntax

```
HRESULT  Eventory(
     [out,  retval]  IInTrustEventory  **ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustEventory**	Log knowledge base.

IInTrustOrganization3

Provides access to an InTrust organization.

Methods

Eventory

Provides access to the log knowledge database associated with the InTrust organization.

Syntax

```
HRESULT  Eventory(
     [out,  retval]  IInTrustEventory  **ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustEventory**	Log knowledge database associated with the InTrust organization.

Name

Returns the name of the InTrust organization.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR	Name of the InTrust organization.

Servers

Provides access to a collection of the InTrust servers in an InTrust organization.

Syntax

```
HRESULT  Servers(
     [out,  retval]  IInTrustServerCollection**  ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustServerCollection**	Collection of InTrust servers.

Scripts

Provides access to a collection of scripts that perform various InTrust operations.

Syntax

```
HRESULT  Scripts(
     [out,  retval]IInTrustScriptCollection**  ppScripts
);
```

Parameters

Name	Type	Meaning
ppScripts	IInTrustScriptCollection**	Collection of InTrust-specific scripts.

Sites

Provides access to a collection of sites in an InTrust organization.

Syntax

```
HRESULT Sites(
    [in, defaultvalue(VisibleSites)] enum SiteCollectionType,
    [out, retval] IInTrustSiteCollection** ppSites
);
```

Parameters

Name	Type	Meaning
	enum SiteCollectionType	Whether you want all sites (including internally-used hidden ones) or just the general-purpose sites.
ppSites	IInTrustSiteCollection**	Collection of InTrust sites.

Repositories2

Provides access to a collection of repositories in an InTrust organization.

Syntax

```
HRESULT Repositories2(
    [out, retval] IInTrustRepositoryCollection2** ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustRepositoryCollection2**	Collection of repositories.

Eventory

Provides access to the organization-wide log knowledge base. See Log Knowledge Base API for details.

Syntax

```
HRESULT Eventory(
    [out, retval] IInTrustEventory **ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustEventory**	Log knowledge base.

IInTrustOrganizationCollection

Provides a collection of all available InTrust organizations.

Methods

Item

Provides access to the specified InTrust organization.

Syntax

```
HRESULT  Item(
     [in]  BSTR  bstrOrganizationIdentity,
     [out,  retval]  IInTrustOrganization**
);
```

Parameters

Name	Type	Meaning
bstrOrganizationIdentity	BSTR	Name of the InTrust organization.
	IInTrustOrganization**	InTrust organization interface.

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT  _NewEnum(
     [out,  retval]  LPUNKNOWN*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Enumerated InTrust organizations.

IInTrustRepository3

Provides the searching and writing capabilities of a repository.

Methods

CustomAttributes

Provides access to the collection (instance of IPropertyCollection) of custom attributes attached to an InTrust repository (instances of IProperty).

Syntax

```
HRESULT CustomAttributes(
    [out, retval] IPropertyCollection** pVal
);
```

Parameter

Name	Type	Meaning
pVal	IPropertyCollection**	Collection of custom attributes attached to the repository.

Description (getter)

Returns the description of the repository.

Syntax

```
HRESULT Description(
    [out, retval] BSTR* description
);
```

Parameter

Name	Type	Meaning
description	BSTR*	Description of the repository.

Description (setter)

Sets the description of the repository.

Syntax

```
HRESULT Description(
    [in] BSTR description
);
```

Parameter

Name	Type	Meaning
description	BSTR	Description of the repository.

ForwardingSettings

Provides access to the forwarding settings for the repository.

Syntax

```
HRESULT ForwardingSettings(
    [out, retval] IForwardingSettings** ppForwardingSettings
);
```

Parameter

Name	Type	Meaning
ppForwardingSettings	IForwardingSettings**	Forwarding settings for the repository.

ID

Returns the GUID of the repository.

Syntax

```
HRESULT ID(
    [out, retval] GUID* pID

);
```

Parameter

Name	Type	Meaning
pID	GUID*	GUID of the repository.

IndexingSettings

Provides access to the indexing settings for the repository.

Syntax

```
HRESULT IndexingSettings(
    [out, retval] IIndexingSettings** ppIndexingSettings
);
```

Parameter

Name	Type	Meaning
ppIndexingSettings	IIndexingSettings**	Indexing settings for the repository.

Inserter

Provides an interface for inserting records into the repository. For details, see Writing Records.

> **CAUTION: A new inserter is created every time you call this method. It's likely that you only want a single unique inserter per repository for all of your writing activity.**

Syntax

```
HRESULT  Inserter(
     [out,  retval]  IRepositoryRecordInserter**  ppInserter
);
```

Parameter

Name	Type	Meaning
ppInserter	IRepositoryRecordInserter**	Record-inserting interface associated with a particular repository.

Name (getter)

Returns the name of the repository.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  name
);
```

Parameter

Name	Type	Meaning
name	BSTR*	Name of the repository. The name is not necessarily unique in an organization.

Name (setter)

Sets the name of the repository.

Syntax

```
HRESULT  Name(
     [in]  BSTR  name
);
```

Parameter

Name	Type	Meaning
name	BSTR	Name of the repository. The name is not necessarily unique in an organization.

Path (getter)

Returns the path to the repository.

Syntax

```
HRESULT Path(
    [out, retval] BSTR* pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR*	UNC path to the share that contains the repository.

Path (setter)

Sets the path to the repository.

Syntax

```
HRESULT Path(
    [in] BSTR path
);
```

Parameter

Name	Type	Meaning
path	BSTR	UNC path to the share that contains the repository.

RepositoryAccessCredentials

Provides access to the credentials that are used for access to the repository.

Syntax

```
HRESULT RepositoryAccessCredentials(
    [in, defaultvalue(CurrentCusomizableCredentials)] enum
CustomizableCredentialsType type,
    [out, retval] ICustomizableCredentials** pCredentials
);
```

Parameters

Name	Type	Meaning
type	enum CustomizableCredentialsType	How repository access is currently configured.
pCredentials	ICustomizableCredentials**	Credentials that are used for access to the repository.

Searcher

Provides an interface for finding records in the repository. For details, see Getting Records.

Syntax

```
HRESULT  Searcher(
     [out,  retval]  IInTrustRepositorySearcher**  ppSearcher
);
```

Parameter

Name	Type	Meaning
ppSearcher	IInTrustRepositorySearcher	A searcher interface that accepts search queries and provides results.

Statuses

Gets the status enumerator for the repository

Syntax

```
HRESULT  Statuses(
     [out,  retval]  IInTrustRepositoryStatusCollection**  pVal
);
```

Parameter

Name	Type	Meaning
pVal	IInTrustRepositoryStatusCollection**	Status enumerator for the repository.

IInTrustRepositoryCollection2

Provides a collection of all repositories available in the InTrust organization.

Methods

Item

Gets the specified repository from a collection.

Syntax

```
HRESULT  Item(
     [in]  BSTR  bstrRepositoryIdentity,
     [out,  retval]  IInTrustRepository3**
);
```

Parameters

Name	Type	Meaning
bstrRepositoryIdentity	BSTR	A piece of information that identifies the repository. You can specify one of the following: • Repository name • Repository GUID • UNC path to the repository share The **Item** method tries to interpret its input parameter as each of these identifiers, in that order.
	IInTrustRepository3**	Repository interface.

Add

Creates a repository with the specified properties in a collection. To configure additional properties, use the methods of the repository that is returned.

Note that even though this makes the repository a member of the collection, it will not actually be created until you have called the Commit method of its IInTrustRepository3 interface.

> **CAUTION: For this operation to succeed, the account you are using must be an InTrust organization administrator. To configure this privilege for the account, do one of the following:**
>
> - **In InTrust Deployment Manager, click Manage | Configure Access.**
> - **In InTrust Manager, open the properties of the root node.**

Syntax

```
HRESULT  Add(
     [in]  BSTR  bstrRepositoryName,
     [in]  BSTR  bstrRepositoryPath,
     [out,  retval]  IInTrustRepository3**
);
```

Parameters

Name	Type	Meaning
bstrRepositoryName	BSTR	Name of the repository.
bstrRepositoryPath	BSTR	UNC path to the share that contains the repository.
	IInTrustRepository3**	Repository interface.

Remove

Removes the specified repository from the collection, deleting it from the InTrust organization configuration.

CAUTION: For this operation to succeed, the account you are using must be an InTrust organization administrator. To configure this privilege for the account, do one of the following:

- **In InTrust Deployment Manager, click Manage | Configure Access.**
- **In InTrust Manager, open the properties of the root node.**

Syntax

```
HRESULT  Remove(
     [in]  BSTR  bstrRepositoryIdentity
);
```

Parameter

Name	Type	Meaning
bstrRepositoryIdentity	BSTR	A piece of information that identifies the repository. You can specify one of the following: • Repository name • Repository GUID • UNC path to the repository share The **Item** method tries to interpret its input parameter as each of these identifiers, in that order.

_NewEnum

References repositories in a collection.

Syntax

```
HRESULT  _NewEnum(
     [out,  retval]  LPUNKNOWN*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Access to repositories in a collection.

IInTrustRepositorySearcher

Provides repository search capabilities.

Method

Search

Runs a repository search using the specified query.

Syntax

```
HRESULT  ID(
     [in]  BSTR  rel_query,
     [out]  IObservable**  search_object
);
```

Parameters

Name	Type	Meaning
rel_query	BSTR	Search query.
search_object	IObservable	Interface that you can subscribe to for search results.

IInTrustScriptCollection

Provides a collection of scripts used in InTrust operations.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT  _NewEnum(
     [out,  retval]  LPUNKNOWN*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Add

Adds a script to the collection.

Syntax

```
HRESULT  Add(
     [out,  retval]  IScript**  ppScript
);
```

Parameters

Name	Type	Meaning
ppScript	IScript**	Script to add to the collection.

Item

Gets a script from the collection by name.

Syntax

```
HRESULT  Item(
     [in]  BSTR  bstrScript,
     [out,  retval]  IScript**  ppScript
);
```

Parameters

Name	Type	Meaning
bstrScript	BSTR	Name of the script.
ppScript	IScript**	The returned script.

Remove

Removes the script with the specified name from the collection.

Syntax

```
HRESULT  Remove(
     [in]  BSTR  bstrScript
);
```

Parameter

Name	Type	Meaning
bstrScript	BSTR	Name of the script to remove.

IInTrustServer

Provides access to an InTrust server.

Methods

Name

Returns the name of the InTrust server.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR*	Name of the InTrust server.

Organization

The InTrust organization that the InTrust server belongs to.

Syntax

```
HRESULT  Organization(
     [out,  retval]  IInTrustOrganization**  pVal
);
```

Parameter

Name	Type	Meaning
pVal	IInTrustOrganization**	InTrust organization interface.

IInTrustServer3

Provides access to an InTrust server.

Methods

ForwardingSupport

Returns the global event forwarding configuration.

Syntax

```
HRESULT  ForwardingSupport(
     [out,  retval]  IInTrustServerForwardingSupport**  ppVal
);
```

Parameter

Name	Type	Meaning
ppVal	IInTrustServerForwardingSupport**	Global event forwarding configuration.

ID

Returns the ID of the InTrust server.

Syntax

```
HRESULT  ID(
     [out,  retval]  BSTR*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR*	ID of the InTrust server.

Name

Returns the name of the InTrust server.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR*	Name of the InTrust server.

Organization

The InTrust organization that the InTrust server belongs to.

Syntax

```
HRESULT Organization(
    [out, retval] IInTrustOrganization** pVal
);
```

Parameter

Name	Type	Meaning
pVal	IInTrustOrganization**	InTrust organization interface.

Version

Returns the InTrust Server version.

Syntax

```
HRESULT Version(
    [out, retval] BSTR* pVersion
);
```

Parameter

Name	Type	Meaning
pVersion	BSTR*	InTrust Server version.

IInTrustServerCollection

Provides a collection of all InTrust servers in the InTrust organization.

Methods

Item

Provides access to the specified InTrust server.

Syntax

```
HRESULT Item(
    [in] BSTR bstrServerIdentity,
    [out, retval] IInTrustServer**
);
```

Parameters

Name	Type	Meaning
bstrServerIdentity	BSTR	Name of the InTrust server.
	IInTrustServer**	InTrust server interface.

_NewEnum

References InTrust servers in a collection.

Syntax

```
HRESULT _NewEnum(
    [out, retval] LPUNKNOWN* pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Enumerated InTrust servers.

IInTrustServerForwardingSupport

Provides access to global InTrust event forwarding settings.

Methods

SupportedMessageFormats

Returns the message formats that InTrust supports for event forwarding.

Syntax

```
HRESULT SupportedMessageFormats(
    [out, retval] IMessageFormatTypeInfoCollection** ppMessageFormats
);
```

Parameter

Name	Type	Meaning
ppMessageFormats	IMessageFormatTypeInfoCollection**	Message formats that InTrust supports for event forwarding.

SupportedTransports

Returns the transports that InTrust event forwarding supports.

Syntax

```
HRESULT  SupportedTransports(
     [out,  retval]  ITransportInfoCollection**  ppTransports
);
```

Parameter

Name	Type	Meaning
ppTransports	ITransportInfoCollection**	Transports that InTrust event forwarding supports.

IInTrustSiteCollection

Represents the sites in an InTrust organization.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT  _NewEnum(
     [out,  retval]  LPUNKNOWN*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Add

Adds a site to the collection.

Syntax

```
HRESULT  Add(
     [in]  enum  SiteType  site,
     [out,  retval]  ISite**
);
```

Parameters

Name	Type	Meaning
site	enum SiteType	What kind of site to add.
	ISite**	The site to add to the collection.

Item

Gets a site from the collection by identity.

Syntax

```
HRESULT  Item(
     [in]  BSTR  siteIdentity,
     [out,  retval]  ISite**
);
```

Parameters

Name	Type	Meaning
siteIdentity	BSTR	Identity of the site. The method accepts site display names and GUIDs.
	ISite**	The returned site.

Remove

Removes the site with the specified identity from the collection.

Syntax

```
HRESULT  Remove(
     [in]  BSTR  siteIdentity
);
```

Parameter

Name	Type	Meaning
siteIdentity	BSTR	Identity of the site to remove. The method accepts site display names and GUIDs.

IIPAddressRangeSiteObject

Represents a range of IP addresses that are included in an InTrust site.

Methods

FromIPAddress (getter)

Returns the starting IP address in the range.

Syntax

```
HRESULT  FromIPAddress(
     [out,  retval]BSTR*  bstrIPAddress
);
```

Parameters

Name	Type	Meaning
bstrIPAddress	BSTR*	Starting IP address in the range.

FromIPAddress (setter)

Sets the starting IP address in the range.

Syntax

```
HRESULT  FromIPAddress(
     [in]BSTR  bstrIPAddress
);
```

Parameters

Name	Type	Meaning
bstrIPAddress	BSTR	Starting IP address in the range.

ToIPAddress (getter)

Returns the ending IP address in the range.

Syntax

```
HRESULT  ToIPAddress(
     [out,  retval]BSTR*  bstrIPAddress
);
```

Parameters

Name	Type	Meaning
bstrIPAddress	BSTR*	Ending IP address in the range.

ToIPAddress (setter)

Sets the ending IP address in the range.

Syntax

```
HRESULT ToIPAddress(
    [in]BSTR bstrIPAddress
);
```

Parameters

Name	Type	Meaning
bstrIPAddress	BSTR	Ending IP address in the range.

IJob2

Represents a subset of the configuration of an InTrust job.

Method

JobCredentials

Provides access to the credentials that the job uses for access to the resources it requires.

Syntax

```
HRESULT JobCredentials(
    [in, defaultvalue(CurrentCusomizableCredentials)] enum
CustomizableCredentialsType type,
    [out, retval] ICustomizableCredentials** ppCredentials
);
```

Parameters

Name	Type	Meaning
type	enum CustomizableCredentialsType	What kind of credential set is used.
ppCredentials	ICustomizableCredentials**	Definition of the credential set used.

IMessageFormatCustomInfo

Represents a customizable script-based message formatter used for event forwarding.

Methods

Data (getter)

Returns the custom script that implements the functionality of the formatter.

Syntax

```
HRESULT  Data(
     [out,  retval]  BSTR*  bstrData
);
```

Parameters

Name	Type	Meaning
bstrData	BSTR*	Custom script that implements the functionality of the formatter.

Data (setter)

Supplies the custom script that implements the functionality of the formatter.

Syntax

```
HRESULT  Data(
     [in]  BSTR  bstrData
);
```

Parameters

Name	Type	Meaning
bstrData	BSTR	Custom script that implements the functionality of the formatter.

IMessageFormatInfo

Provides access to the formatting configuration for forwarded events.

Methods

Name	Type	Meaning

Name	Type	Meaning

MessageFormatType

Returns the configuration of message format.

Syntax

```
HRESULT  MessageFormatType(
    [out,  retval]  IMessageFormatTypeInfo**  ppMessageFormatTypeInfo
);
```

Parameter

Name	Type	Meaning
ppMessageFormatTypeInfo	IMessageFormatTypeInfo**	Configuration of the message format.

Name

Returns the display name of the message format.

Syntax

```
HRESULT  Name(
    [out,  retval]  BSTR*  val
);
```

Parameter

Name	Type	Meaning
val	BSTR*	Display name of the message format.

IMessageFormatTypeInfo

Defines a message format for forwarded events and can be used as a template for creating new formats.

Methods

CreateMessageFormat

Returns a new message format item.

Syntax

```
HRESULT  CreateMessageFormat(
     [out,  retval]IMessageFormatInfo**  ppMessageFormat
);
```

Parameters

Name	Type	Meaning
ppMessageFormat	IMessageFormatInfo**	New message format for forwarded events.

GUID

Returns the GUID of the message format.

Syntax

```
HRESULT  GUID(
     [out,  retval]  BSTR*  GUID
);
```

Parameters

Name	Type	Meaning
GUID	BSTR*	GUID of the message format.

IsCustomizable

Returns whether this is a customizable script-based message format.

Syntax

```
HRESULT  IsCustomizable(
     [out,  retval]  VARIANT_BOOL*  val
);
```

Parameters

Name	Type	Meaning
val	VARIANT_BOOL*	Whether this is a customizable script-based message format.

Name

Returns the display name of the message format.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  val
);
```

Parameters

Name	Type	Meaning
val	BSTR*	Display name of the message format.

IMessageFormatTypeInfoCollection

Provides a collection of all message format types supported by InTrust event forwarding.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT  _NewEnum(
     [out,  retval]  LPUNKNOWN*  pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Item

Gets a message format type from the collection by GUID.

Syntax

```
HRESULT  Item(
     [in]  BSTR  GUID,
     [out,  retval]  IMessageFormatTypeInfo**
);
```

Parameters

Name	Type	Meaning
GUID	BSTR	GUID of the message format type.
	IMessageFormatTypeInfo**	The message format type.

IMicrosoftNetworkSite

Represents an InTrust site of the Microsoft Windows Network type.

Methods

AgentAccessCredentials

Provides access to the credentials used for running the agent service.

Syntax

```
HRESULT  AgentAccessCredentials(
     [in,  defaultvalue(CurrentCusomizableCredentials)]  enum
CustomizableCredentialsType,
     [out,  retval]  ICustomizableCredentials**  pCredentials
);
```

Parameters

Name	Type	Meaning
	enum CustomizableCredentialsType	What kind of credential set to use.
pCredentials	ICustomizableCredentials**	Credential set to use.

DomainEnumeration

Provides access to the choice of domain enumeration method for sites to use.

Syntax

```
HRESULT DomainEnumeration(
    [in, defaultvalue(CurrentDomainEnumeration)] enum DomainEnumerationType,
    [out, retval] IDomainEnumeration** pDomainEnumeration
);
```

Parameters

Name	Type	Meaning
	enum DomainEnumerationType	What kind of domain enumeration method to use.
pDomainEnumeration	IDomainEnumeration**	Choice of domain enumeration mechanism.

InstallAgentsAutomatically (getter)

Returns whether automatic installation of agents on the site computers is enabled.

Syntax

```
HRESULT InstallAgentsAutomatically(
    [out, retval] VARIANT_BOOL* pInstallAutomatically
);
```

Parameter

Name	Type	Meaning
pInstallAutomatically	VARIANT_BOOL*	Whether automatic installation of agents on the site computers is enabled.

InstallAgentsAutomatically (setter)

Sets whether automatic installation of agents on the site computers is enabled.

Syntax

```
HRESULT InstallAgentsAutomatically(
    [in] VARIANT_BOOL pInstallAutomatically
);
```

Parameter

Name	Type	Meaning
pInstallAutomatically	VARIANT_BOOL	Whether automatic installation of agents on the site computers is enabled.

SiteAccessCredentials

Provides access to the credentials used for site enumeration and installation of agents on site computers.

Syntax

```
HRESULT SiteAccessCredentials(
    [in, defaultvalue(CurrentCusomizableCredentials)] enum
CustomizableCredentialsType,
    [out, retval] ICustomizableCredentials** pCredentials
);
```

Parameters

Name	Type	Meaning
	enum CustomizableCredentialsType	What kind of credential set to use.
pCredentials	ICustomizableCredentials**	Credential set to use.

IMultiRepositorySearcher

A container for search objects that lets you search in all of the specified repositories simultaneously.

> **CAUTION: This container is optimized for shared use. Therefore, it is strongly recommended that you create only one IMultiRepositorySearcher and reuse it rather than creating different IMultiRepositorySearcher instances for different search queries and sets of repositories.**

Methods

MakeMultiSearchObject

Creates a search object that uses multiple repositories at once.

Syntax

```
HRESULT MakeMultiSearchObject(
    [in] BSTR rel_query,
    [in] SAFEARRAY(IDispatch) psaSeachers,
    [out, retval] IObservable** search_object

);
```

Parameters

Name	Type	Meaning
rel_query	BSTR	Search query.
psaSeachers	SAFEARRAY(IDispatch)	Searcher interfaces for the repositories you want to search in.
search_object	IObservable**	Interface that provides search functionality.

IMultiRepositorySearcherFactory

Creates an instance of IMultiRepositorySearcher.

CreateMultiRepositorySearcher

Creates the IMultiRepositorySearcher.

Syntax

```
HRESULT CreateMultiRepositorySearcher(
    [in] VARIANT eventory_xml,
    [out, retval] IMultiRepositorySearcher** rep_searcher
);
```

Parameters

Name	Type	Meaning
eventory_xml	VARIANT	String representation of the log knowledge base to use with the multi-repository searches. To use the fallback knowledge base, specify **null**.
rep_searcher	IMultiRepositorySearcher **	The searcher interface capable of working with multiple repositories at once.

IObservable

Defines a provider for push-based notification.

Method

Subscribe

Syntax

```
HRESULT Subscribe(
    [in] IObserver* observer,
    [out] ICookie** cookie
);
```

Parameters

Name	Type	Meaning
observer	IObserver*	Source of push-based notifications.
cookie	ICookie**	Keeps the search active while present.

IObserver

Provides a mechanism for receiving push-based notifications. You need to create your own implementation of this interface.

Methods

OnDone

Notifies the observer that the provider has finished sending push-based notifications.

Syntax

```
void OnDone();
```

OnError

Notifies the observer that the provider has experienced an error condition.

Syntax

```
void OnError(
    [in] HRESULT hr,
    [in] BSTR description
);
```

Parameters

Name	Type	Meaning
hr	HRESULT	Operation result.
description	BSTR	Additional information about the error.

OnNext

Provides the observer with new data.

Syntax

```
void OnNext(
    [in] IUnknown* data
);
```

Parameter

Name	Type	Meaning
data	IUnknown*	The current notification information.

IOrganizationalUnitSiteObject

Represents an organizational unit from which to put computers in an InTrust site.

Methods

CanonicalName (getter)

Returns the canonical name of the organizational unit.

Syntax

```
HRESULT CanonicalName(
    [out, retval]BSTR* bstrCanonicalName
);
```

Parameter

Name	Type	Meaning
bstrCanonicalName	BSTR*	Canonical name of the organizational unit.

CanonicalName (setter)

Sets the canonical name of the organizational unit.

Syntax

```
HRESULT CanonicalName(
    [in]BSTR bstrCanonicalName
);
```

Parameter

Name	Type	Meaning
bstrCanonicalName	BSTR	Canonical name of the organizational unit.

DistinguishedName (getter)

Returns the distinguished name of the organizational unit.

Syntax

```
HRESULT  DistinguishedName(
     [out,  retval]BSTR*  bstrDistinguishedName
);
```

Parameter

Name	Type	Meaning
bstrDistinguishedName	BSTR*	Distinguished name of the organizational unit.

DistinguishedName (setter)

Sets the distinguished name of the organizational unit.

Syntax

```
HRESULT  DistinguishedName(
     [in]BSTR  bstrDistinguishedName
);
```

Parameter

Name	Type	Meaning
bstrDistinguishedName	BSTR	Distinguished name of the organizational unit.

Domain (getter)

Returns the domain of the organizational unit.

Syntax

```
HRESULT  Domain(
     [out,  retval]BSTR*  bstrDomain
);
```

Parameter

Name	Type	Meaning
bstrDomain	BSTR*	Domain of the organizational unit.

Domain (setter)

Sets the domain of the organizational unit.

Syntax

```
HRESULT  Domain(
     [in]BSTR  bstrDomain
);
```

Parameter

Name	Type	Meaning
bstrDomain	BSTR	Domain of the organizational unit.

IProperty

Represents a property attached to an InTrust repository. A property is a way to tag repositories for arbitrary purposes.

Methods

PropertyName (setter)

Sets the name of the property.

Syntax

```
HRESULT  PropertyName(
     [in]  BSTR  pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR	Name of the property. The name must be unique in the property collection (see IPropertyCollection).

PropertyValue (getter)

Returns the value of the property.

Syntax

```
HRESULT  PropertyValue(
     [out,  retval]  VARIANT  *pVal
);
```

Parameter

Name	Type	Meaning
pVal	VARIANT*	Value of the property.

PropertyValue (setter)

Sets the value of the property.

Syntax

```
HRESULT  PropertyValue(
     [in]  VARIANT  pVal
);
```

Parameter

Name	Type	Meaning
pVal	VARIANT	Value of the property.

PropertyName

Returns the name of the property.

> **NOTE:** There is no setter method for the name of a property. Instead of renaming an existing property, you need to create a new one in the property collection (IPropertyCollection) and assign it the value you need. The old property can be deleted using the collection's **Remove** method.

Syntax

```
HRESULT  PropertyName(
     [out,  retval]  BSTR  *pVal
);
```

Parameter

Name	Type	Meaning
pVal	BSTR*	Name of the property.

IPropertyCollection

Represents a collection of properties associated with an InTrust repository. Access to the collections is gained through specialized methods of the IInTrustRepository3 interface (such as **CustomAttributes** and **ForwardingProperties**), which filter the available properties by purpose.

Methods

Item

Gets a property from the collection by name.

Syntax

```
HRESULT  Item(
     [in]  BSTR  bstrPropertyName,
     [out,  retval]  IProperty**  ppProperty
);
```

Parameters

Name	Type	Meaning
bstrPropertyName	BSTR	Name of the property. This name must exist in the collection.
ppProperty	IProperty**	The property.

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT  _NewEnum(
     [out,  retval]  LPUNKNOWN*  pVal
);
```

Parameters

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Set

Sets the specified property if it exists or creates it if it doesn't.

Syntax

```
HRESULT  Set(
  [in]  BSTR  bstrPropertyName,
     [in]  VARIANT  varPropertyValue
);
```

Parameters

Name	Type	Meaning
bstrPropertyName	BSTR	Name of the property to add. The name must be unique.
varPropertyValue	BSTR	The value to set.

Remove

Removes a property from the collection by name.

Syntax

```
HRESULT  Remove(
     [in]  BSTR  bstrPropertyName
);
```

Parameter

Name	Type	Meaning
bstrPropertyName	BSTR	Name of the property to remove.

IRepositoryRecordInserter

Provides write access to the repository that it is associated with and manages one or more IRepositoryRecordInserterLight interfaces, which do the actual writing. For each IRepositoryRecordInserterLight, it also stores predefined field values that are the same in all records written by that IRepositoryRecordInserterLight.

Incoming records are pushed to the repository at regular intervals. However, you can force an immediate write by calling the **Commit** method.

Methods

BindFields

Sets the values of the path-specifying fields for records that will be written to the same repository file.

Syntax

```
HRESULT BindFields(
```

```
    [in] tags path,

    [out, retval] IRepositoryRecordInserterLight**

);
```

Parameters

Name	Type	Meaning
path	tags	The field record values that you specify here are supposed to be the same for all records generated by the IRepositoryRecordInserterLight that will be initialized.
	IRepositoryRecordInserterLight **	Record-inserting interface with some record field values predefined.

PutRecords

Writes the specified records to the repository asynchronously.

Syntax

```
HRESULT PutRecords(

    [in] SAFEARRAY(struct record) records

);
```

Parameter

Name	Type	Meaning
records	SAFEARRAY(struct record)	Records to put in the repository.

PutRecords2

Writes the specified records to the repository asynchronously. This method is similar to **PutRecords**, except the type of the input parameter. Using the IBulkRecord interface for input makes it possible to write event records converted by IEventToRecordFormatter. For details, see Event Record Data Structures.

Syntax

```
HRESULT PutRecords2(

    [in] IBulkRecord* pBulkRecord

);
```

Parameter

Name	Type	Meaning
records	IBulkRecord* pBulkRecord	Records (normally, converted from events) to put in the repository.

Commit

Performs all deferred record writes synchronously.

> **CAUTION: This operation is resource-intensive and should not be used needlessly. For example, committing after each record is strongly discouraged. InTrust commits records automatically every 60 seconds.**
>
> **Forcing a commit is acceptable in situations like the following:**
>
> - **You need to confirm that a batch of events or records has safely arrived in the repository.**
> - **You are writing events or records out of order. See the corresponding note in Writing Events.**

Syntax

```
HRESULT Commit();
```

Return Values

Name	Value	Meaning
ITRT_E_COMMIT_TO_INTERMEDIATE_STORE_FAILED	0x8ADD1002	Cannot commit records to intermediate store on InTrust server.
ITRT_E_COMMIT_TO_REPOSITORY_FAILED	0x8ADD1003	Cannot commit records to repository.
ITRT_E_COMMIT_IN_PROGRESS	0x8ADD1004	Commit is still in progress.

IRepositoryRecordInserter2

Provides write access to the repository that it is associated with and manages one or more IRepositoryRecordInserterLight interfaces, which do the actual writing. For each IRepositoryRecordInserterLight, it also stores predefined field values that are the same in all records written by that IRepositoryRecordInserterLight.

Incoming records are pushed to the repository at regular intervals. However, you can force an immediate write by calling the **Commit** method.

Methods

BindFields

Sets the values of the path-specifying fields for records that will be written to the same repository file.

Syntax

```
HRESULT BindFields(

    [in] tags path,

    [out, retval] IRepositoryRecordInserterLight**

);
```

Parameters

Name	Type	Meaning
path	tags	The field record values that you specify here are supposed to be the same for all records generated by the IRepositoryRecordInserterLight that will be initialized.
	IRepositoryRecordInserterLight **	Record-inserting interface with some record field values predefined.

PutRecords

Writes the specified records to the repository asynchronously.

Syntax

```
HRESULT PutRecords(

    [in] SAFEARRAY(struct record) records

);
```

Parameter

Name	Type	Meaning
records	SAFEARRAY(struct record)	Records to put in the repository.

PutRecords2

Writes the specified records to the repository asynchronously. This method is similar to **PutRecords**, except the type of the input parameter. Using the IBulkRecord interface for input makes it possible to write event records converted by IEventToRecordFormatter. For details, see Event Record Data Structures.

Syntax

```
HRESULT PutRecords2(

    [in] IBulkRecord* pBulkRecord

);
```

Parameter

Name	Type	Meaning
records	IBulkRecord* pBulkRecord	Records (normally, converted from events) to put in the repository.

Commit

Performs all deferred record writes synchronously.

> **CAUTION: This operation is resource-intensive and should not be used needlessly. For example, committing after each record is strongly discouraged. InTrust commits records automatically every 60 seconds.**
>
> **Forcing a commit is acceptable in situations like the following:**
>
> - **You need to confirm that a batch of events or records has safely arrived in the repository.**
> - **You are writing events or records out of order. See the corresponding note in Writing Events.**

Syntax

```
HRESULT Commit();
```

Return Values

Name	Value	Meaning
ITRT_E_COMMIT_TO_INTERMEDIATE_STORE_FAILED	0x8ADD1002	Cannot commit records to intermediate store on InTrust server.
ITRT_E_COMMIT_TO_REPOSITORY_FAILED	0x8ADD1003	Cannot commit records to repository.
ITRT_E_COMMIT_IN_PROGRESS	0x8ADD1004	Commit is still in progress.

Commit2

Performs deferred record writes. This method either submits records to a queue on the server or puts them directly in the repository, depending on the argument. Using **RepositoryCommitType::ToRepositoryRepositoryCommitType** as the argument is equivalent to calling the **Commit** method.

> **CAUTION: Direct writes to the repository are resource-intensive and should not be used needlessly. For example, committing after each record is strongly discouraged. InTrust commits records automatically every 60 seconds.**
>
> **Forcing a commit is acceptable in situations like the following:**
>
> - **You need to confirm that a batch of events or records has safely arrived in the repository.**
> - **You are writing events or records out of order. See the corresponding note in Writing Events.**

Syntax

```
HRESULT Commit2(
    [in, defaultvalue(ToRepositoryRepositoryCommitType)] RepositoryCommitType
```

```
commitType
);
```

Parameter

Name	Type	Meaning
commitType	RepositoryCommitType	Whether to queue the committed records on the server or put them directly in the repository.

Return Values if RepositoryCommitType::ToRepositoryRepositoryCommitType is Used

Name	Value	Meaning
ITRT_E_COMMIT_TO_INTERMEDIATE_STORE_FAILED	0x8ADD1002	Cannot commit records to intermediate store on InTrust server.
ITRT_E_COMMIT_TO_REPOSITORY_FAILED	0x8ADD1003	Cannot commit records to repository.
ITRT_E_COMMIT_IN_PROGRESS	0x8ADD1004	Commit is still in progress.

IRepositoryRecordInserterLight

Generates valid record structures from predefined and significant values and writes them to the repository.

> **NOTE:IRepositoryRecordInserterLight or IRepositoryRecordInserter: when to use which?**
> Use IRepositoryRecordInserterLight if you need to write large numbers of records with coinciding values in specific fields. Otherwise, using IRepositoryRecordInserter should be more efficient.

Method

PutRecords

Syntax

```
HRESULT PutRecords(
    [in] SAFEARRAY(struct contents) recordFields
);
```

Parameter

Name	Type	Meaning
recordFields	SAFEARRAY (struct contents)	Field value structures to convert to records. Missing fields will be filled in based on the tags structure instance associated with this IRepositoryRecordInserterLight.

IScript

Represents a script used in InTrust operations.

Methods

Description (getter)

Returns the description of the script.

Syntax

```
HRESULT Description(
    [out, retval] BSTR* description
);
```

Parameter

Name	Type	Meaning
description	BSTR*	Description of the script.

Description (setter)

Sets the description of the script.

Syntax

```
HRESULT Description(
    [in] BSTR description
);
```

Parameter

Name	Type	Meaning
description	BSTR	Description of the script.

ID

Returns the ID of the script.

Syntax

```
HRESULT ID(
    [out, retval] BSTR* pID
);
```

Parameter

Name	Type	Meaning
pID	BSTR*	ID of the script.

Language (getter)

Returns which language the script is in.

Syntax

```
HRESULT Language(
    [out, retval] enum ScriptLanguage* language
);
```

Parameter

Name	Type	Meaning
language	enum ScriptLanguage*	Which language the script is in.

Language (setter)

Sets which language the script is in.

Syntax

```
HRESULT Language(
    [in] enum ScriptLanguage language
);
```

Parameter

Name	Type	Meaning
language	enum ScriptLanguage	Which language the script is in.

Name (getter)

Returns the name of the script.

Syntax

```
HRESULT Name(
    [out, retval] BSTR *name
);
```

Parameter

Name	Type	Meaning
name	BSTR*	Name of the script.

Name (setter)

Sets the name of the script.

Syntax

```
HRESULT  Name(
     [in]  BSTR  name
);
```

Parameter

Name	Type	Meaning
name	BSTR	Name of the script.

Parameters

Provides access to a collection of the script's parameters.

Syntax

```
HRESULT  Parameters(
     [out,  retval]  IScriptParameterCollection**  ppParameters
);
```

Parameter

Name	Type	Meaning
ppParameters	IScriptParameterCollection**	Collection of the script's parameters.

Script (getter)

Returns the script source code.

Syntax

```
HRESULT  Script(
     [out,  retval]  BSTR*  script
);
```

Parameter

Name	Type	Meaning
script	BSTR*	Script source code.

Script (setter)

Overwrites the script source code.

Syntax

```
HRESULT  Script(
     [in]  BSTR  script
);
```

Parameters

Name	Type	Meaning
script	BSTR	Script source code.

IScriptArgument

Represents an argument used with an InTrust site enumeration script.

Methods

Description

Returns the description of the argument.

Syntax

```
HRESULT  Description(
     [out,  retval]  BSTR*  description
);
```

Parameter

Name	Type	Meaning
description	BSTR*	Description of the argument.

ID

Returns the ID of the argument.

Syntax

```
HRESULT ID(
    [out, retval] BSTR* pID
);
```

Parameter

Name	Type	Meaning
pID	BSTR*	ID of the argument.

IsCustom

Returns whether this is a custom argument.

Syntax

```
HRESULT IsCustom(
    [out, retval] VARIANT_BOOL* bIsCustom
);
```

Parameter

Name	Type	Meaning
bIsCustom	VARIANT_BOOL*	Whether this is a custom argument.

Name

Returns the name of the argument.

Syntax

```
HRESULT Name(
    [out, retval] BSTR *name
);
```

Parameter

Name	Type	Meaning
name	BSTR*	Name of the argument.

Value (getter)

Returns the value of the argument.

Syntax

```
HRESULT Value(
    [out, retval] BSTR* value
);
```

Parameter

Name	Type	Meaning
value	BSTR*	Value of the argument.

Value (setter)

Sets the value of the argument.

Syntax

```
HRESULT Value(
    [in] BSTR value
);
```

Parameter

Name	Type	Meaning
value	BSTR	Value of the argument.

IScriptArgumentCollection

Represents the arguments defined for an InTrust script.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT _NewEnum(
    [out, retval] LPUNKNOWN* pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Item

Gets an argument from the collection by name.

Syntax

```
HRESULT  Item(
     [in]  BSTR  bstrArgumentName,
     [out,  retval]  IScriptArgument**  ppArgument
);
```

Parameters

Name	Type	Meaning
bstrArgumentName	BSTR	Name of the argument
ppArgument	IScriptArgument**	The returned argument.

IScriptParameter

Represents a customizable parameter defined for an InTrust script.

Methods

DefaultValue (getter)

Returns the default value of the parameter.

Syntax

```
HRESULT  DefaultValue(
     [out,  retval]  BSTR*  bstrDefaultValue
);
```

Parameter

Name	Type	Meaning
bstrDefaultValue	BSTR*	Default value of the parameter.

DefaultValue (setter)

Sets the default value for the parameter.

Syntax

```
HRESULT DefaultValue(
    [in] BSTR bstrDefaultValue
);
```

Parameter

Name	Type	Meaning
bstrDefaultValue	BSTR	Default value of the parameter.

Description (getter)

Returns the description of the parameter.

Syntax

```
HRESULT Description(
    [out, retval] BSTR* bstrDescription
);
```

Parameter

Name	Type	Meaning
bstrDescription	BSTR*	Description of the parameter.

Description (setter)

Sets the description of the parameter.

Syntax

```
HRESULT Description(
    [in] BSTR bstrDescription
);
```

Parameter

Name	Type	Meaning
bstrDescription	BSTR	Description of the parameter.

ID

Returns the ID of the script parameter.

Syntax

```
HRESULT ID(
    [out, retval] BSTR* bstrID
);
```

Parameter

Name	Type	Meaning
bstrID	BSTR*	ID of the script parameter.

Name (getter)

Returns the name of the parameter.

Syntax

```
HRESULT Name(
    [out, retval] BSTR* bstrName
);
```

Parameter

Name	Type	Meaning
bstrName	BSTR*	Name of the parameter.

Name (setter)

Sets the name of the parameter.

Syntax

```
HRESULT Name(
    [in] BSTR bstrName
);
```

Parameter

Name	Type	Meaning
bstrName	BSTR	Name of the parameter.

IScriptParameterCollection

Represents the parameters defined for an InTrust script.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT _NewEnum(
    [out, retval] LPUNKNOWN* pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Add

Adds a parameter to the collection.

Syntax

```
HRESULT Add(
    [out, retval] IScriptParameter** ppParameter
);
```

Parameters

Name	Type	Meaning
ppParameter	IScriptParameter**	Parameter to add to the collection.

Item

Gets a parameter from the collection by name.

Syntax

```
HRESULT Item(
    [in] BSTR bstrParameterName,
    [out, retval] IScriptParameter** ppParameter
);
```

Parameters

Name	Type	Meaning
bstrParameterName	BSTR	Name of the parameter.

Name	Type	Meaning
ppParameter	IScriptParameter**	The returned parameter.

Remove

Removes the parameter with the specified name from the collection.

Syntax

```
HRESULT  Remove(
     [in]  BSTR  bstrParameterName
);
```

Parameter

Name	Type	Meaning
bstrParameterName	BSTR	Name of the parameter to remove.

ISite

Represents an InTrust site, which can be a regular site visible in InTrust Manager or a hidden internal site associated with a collection visible in InTrust Deployment Manager.

To work with methods that are specific to sites of the Microsoft Windows Network type, cast this to IMicrosoftNetworkSite

Methods

Description (getter)

Returns the description of the site.

Syntax

```
HRESULT  Description(
     [out,  retval]BSTR*  bstrDescription
);
```

Parameter

Name	Type	Meaning
bstrDescription	BSTR*	Description of the site.

Description (setter)

Sets the description of the site.

Syntax

```
HRESULT Description(
    [in]BSTR bstrDescription
);
```

Parameter

Name	Type	Meaning
bstrDescription	BSTR	Description of the site.

EnumerationPeriod (getter)

Returns the interval (in hours) between site enumerations that refresh the site membership.

Syntax

```
HRESULT EnumerationPeriod(
    [out, retval] long* pEnumerationPeriod
);
```

Parameter

Name	Type	Meaning
pEnumerationPeriod	long*	Interval (in hours) between site enumerations that refresh the site membership.

EnumerationPeriod (setter)

Sets the interval (in hours) between site enumerations that refresh the site membership.

Syntax

```
HRESULT EnumerationPeriod(
    [in] long enumerationPeriod
);
```

Parameter

Name	Type	Meaning
pEnumerationPeriod	long	Interval (in hours) between site enumerations that refresh the site membership.

ID

Returns the ID of the site.

Syntax

```
HRESULT ID(
    [out, retval] BSTR* pID
);
```

Parameter

Name	Type	Meaning
pID	BSTR*	ID of the site.

Name (getter)

Returns the name of the site.

Syntax

```
HRESULT Name(
    [out, retval] BSTR* bstrName
);
```

Parameter

Name	Type	Meaning
bstrName	BSTR*	Name of the site.

Name (setter)

Sets the name of the site.

Syntax

```
HRESULT Name(
    [in]BSTR bstrName
);
```

Parameter

Name	Type	Meaning
bstrName	BSTR	Name of the site.

Server (getter)

Returns the InTrust server that manages the site.

Syntax

```
HRESULT  Server(
     [out,  retval]  IInTrustServer3**  ppServer
);
```

Parameter

Name	Type	Meaning
ppServer	IInTrustServer3**	InTrust server that manages the site.

Server (setter)

Sets the InTrust server that manages the site.

Syntax

```
HRESULT  Server(
     [in]  IInTrustServer3*  pServer
);
```

Parameter

Name	Type	Meaning
pServer	IInTrustServer3*	InTrust server that manages the site.

SiteObjects

Provides access to the collection of objects in the site.

Syntax

```
HRESULT  SiteObjects(
     [out,  retval]  ISiteObjectCollection**  pSiteObjects
);
```

Parameter

Name	Type	Meaning
pSiteObjects	ISiteObjectCollection**	Collection of objects in the site.

Type

Returns the type of the site.

Syntax

```
HRESULT  Type(
     [out,  retval]  enum  SiteType*  pSiteType
);
```

Parameter

Name	Type	Meaning
pSiteType	enum SiteType*	Type of the site.

ISiteComputer

Represents a computer that is included in an InTrust site.

Methods

AccessName

Returns the access name of the computer. This is either the IP address or the same as the name returned by the **OriginalName** method.

Syntax

```
HRESULT  AccessName(
     [out,  retval]  BSTR*  pbstrName
);
```

Parameter

Name	Type	Meaning
pbstrName	BSTR*	Access name of the computer.

AgentID

Returns the ID of the agent installed on the computer.

Syntax

```
HRESULT  AgentID(
     [out,  retval]  BSTR*  pbstrAgentID
);
```

Parameter

Name	Type	Meaning
pbstrAgentID	BSTR*	ID of the agent installed on the computer.

Alive

Returns whether the computer is treated as active by the InTrust server.

Syntax

```
HRESULT  Alive(
     [out,  retval]  VARIANT_BOOL*  pvbAlive
);
```

Parameter

Name	Type	Meaning
pvbAlive	VARIANT_BOOL*	Whether the computer is treated as active by the InTrust server.

IPAddress

Returns the IP address of the computer.

Syntax

```
HRESULT  IPAddress(
     [out,  retval]  long*  lIP
);
```

Parameter

Name	Type	Meaning
lIP	long*	IP address of the computer.

Name

Returns the name of the computer, as shown, for example, in the system properties in Windows.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  pbstrName
);
```

Parameter

Name	Type	Meaning
pbstrName	BSTR*	Name of the computer.

OfficialHostName

Returns the official host name of the computer. This is either the FQDN or the canonical name, whichever is the result of resolving the name.

Syntax

```
HRESULT  OfficialHostName(
     [out,  retval]  BSTR*  pbstrName
);
```

Parameter

Name	Type	Meaning
pbstrName	BSTR*	Official host name of the computer.

OriginalName

Returns the original name of the computer. This is the name that was originally provided by the user for adding the computer to the site.

Syntax

```
HRESULT  OriginalName(
     [out,  retval]  BSTR*  pbstrName
);
```

Parameter

Name	Type	Meaning
pbstrName	BSTR*	Original name of the computer.

Status

Returns the current status of the computer.

Syntax

```
HRESULT  Status(
     [out,  retval]  BSTR*  pbstrName
);
```

Parameter

Name	Type	Meaning
pbstrName	BSTR*	Current status of the computer.

ISiteIndexBuilder

Represents the distributed indexing configuration for a repository.

Methods

IndexAccess

Provides access to the security settings for distributed indexing of the repository.

Syntax

```
HRESULT  IndexBuilderAccessCredentials(
    [in,  defaultvalue(CurrentCusomizableCredentials)]  enum
CustomizableCredentialsType,
    [out,  retval]  ICustomizableCredentials**  pCredentials
);
```

Parameters

Name	Type	Meaning
	enum CustomizableCredentialsType	
pCredentials	ICustomizableCredentials**	Security settings for distributed indexing of the repository.

SiteId (getter)

Returns the ID of the InTrust site whose agents must perform distributed indexing operations.

Syntax

```
HRESULT  SiteId(
    [out,  retval]  BSTR*  siteId
);
```

Parameter

Name	Type	Meaning
siteId	BSTR*	ID of the InTrust site whose agents must perform distributed indexing operations.

SiteId (setter)

Specifies the ID of the InTrust site whose agents must perform distributed indexing operations.

Syntax

```
HRESULT SiteId(
    [in] BSTR siteId
);
```

Parameter

Name	Type	Meaning
siteId	BSTR	ID of the InTrust site whose agents must perform distributed indexing operations.

IndexBuilderAccess

Provides access to the security settings for performing repository indexing.

Syntax

```
HRESULT IndexBuilderAccess(
    [in, defaultvalue(CurrentIndexBuilderAccess)] enum IndexBuilderAccessType,
    [out, retval] IIndexBuilderAccess** pIndexPathType
);
```

Parameters

Name	Type	Meaning
	enum IndexBuilderAccessType	What kind of account is used for repository indexing.
pIndexPathType	IIndexBuilderAccess**	Security settings for performing repository indexing.

ISiteObject

Represents a computer-specifying object that can be included in an InTrust site. Sites can be populated by indicating computers in a variety of ways, including IP ranges and Active DIrectory domains.

Methods

Type

Returns the type of the site object.

Syntax

```
HRESULT  Type(
     [out,  retval]enum  SiteObjectType*  type
);
```

Parameter

Name	Type	Meaning
type	enum SiteObjectType*	Type of the site object.

ID

Returns the ID of the site object.

Syntax

```
HRESULT  ID(
     [out,  retval]  BSTR*  pID
);
```

Parameter

Name	Type	Meaning
pID	BSTR*	ID of the site object.

ISiteObjectCollection

Represents the computer-specifying objects included in a site. Sites can be populated by indicating computers in a variety of ways, including IP ranges and Active DIrectory domains.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT _NewEnum(
    [out, retval] LPUNKNOWN* pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Add

Adds a site object to the collection.

Syntax

```
HRESULT Add(
    [in] enum SiteObjectType objectType,
    [out, retval] ISiteObject** ppSiteObject
);
```

Parameters

Name	Type	Meaning
objectType	enum SiteObjectType	What kind of site object to add.
ppSiteObject	ISiteObject**	The site object to add to the collection.

Item

Gets a site object from the collection by ID.

Syntax

```
HRESULT Item(
    [in] BSTR siteObjectId,
    [out, retval] ISiteObject** ppSiteObject
);
```

Parameters

Name	Type	Meaning
siteObjectId	BSTR	ID of the site object to get.
ppSiteObject	ISiteObject**	The returned site object.

Remove

Removes the specified site object from the collection.

Syntax

```
HRESULT  Remove(
     [in]  BSTR  bstrSiteObject
);
```

Parameter

Name	Type	Meaning
bstrSiteObject	BSTR	The site object to remove.

ITask2

Represents a subset of the configuration of an InTrust scheduled task.

Method

TaskCredentials

Provides access to the credentials used for running the task.

Syntax

```
HRESULT  TaskCredentials(
     [in,  defaultvalue(CurrentCusomizableCredentials)]  enum
CustomizableCredentialsType  type,
     [out,  retval]  ICustomizableCredentials**  ppCredentials
);
```

Parameters

Name	Type	Meaning
type	enum CustomizableCredentialsType	What kind of credential set is used.
ppCredentials	ICustomizableCredentials**	Credentials for running the task.

ITransportInfo

Represents a transport type supported by InTrust event forwarding.

Methods

GUID

Returns the GUID of the transport type.

Syntax

```
HRESULT  GUID(
     [out,  retval]  BSTR*  GUID
);
```

Parameter

Name	Type	Meaning
GUID	BSTR*	GUID of the transport type.

Name

Returns the display name of the transport type.

Syntax

```
HRESULT  Name(
     [out,  retval]  BSTR*  val
);
```

Parameter

Name	Type	Meaning
val	BSTR*	Display name of the transport type.

ITransportInfoCollection

Provides a collection of all transport types supported by InTrust event forwarding.

Methods

_NewEnum

Returns an enumerator for the collection.

Syntax

```
HRESULT _NewEnum(
    [out, retval] LPUNKNOWN* pVal
);
```

Parameter

Name	Type	Meaning
pVal	LPUNKNOWN*	Collection enumerator.

Item

Gets a transport type from the collection by GUID.

Syntax

```
HRESULT Item(
    [in] BSTR GUID,
    [out, retval] ITransportInfo**
);
```

Parameters

Name	Type	Meaning
GUID	BSTR	GUID of the transport type.
	ITransportInfo**	The transport type.

We are more than just a name

We are on a quest to make your information technology work harder for you. That is why we build community-driven software solutions that help you spend less time on IT administration and more time on business innovation. We help you modernize your data center, get you to the cloud quicker and provide the expertise, security and accessibility you need to grow your data-driven business. Combined with Quest's invitation to the global community to be a part of its innovation, and our firm commitment to ensuring customer satisfaction, we continue to deliver solutions that have a real impact on our customers today and leave a legacy we are proud of. We are challenging the status quo by transforming into a new software company. And as your partner, we work tirelessly to make sure your information technology is designed for you and by you. This is our mission, and we are in this together. Welcome to a new Quest. You are invited to Join the Innovation™.

Our brand, our vision. Together.

Our logo reflects our story: innovation, community and support. An important part of this story begins with the letter Q. It is a perfect circle, representing our commitment to technological precision and strength. The space in the Q itself symbolizes our need to add the missing piece — you — to the community, to the new Quest.

Contacting Quest

For sales or other inquiries, visit www.quest.com/contact.

Technical support resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request
- View Knowledge Base articles
- Sign up for product notifications
- Download software and technical documentation
- View how-to-videos
- Engage in community discussions
- Chat with support engineers online
- View services to assist you with your product