PureConnect®

2020 R4

Generated:

09-November-2020

Content last updated:

21-June-2019

See Change Log for summary of changes.

Introduction to IceLib

Technical Reference

Abstract

This document introduces the Interaction Center Extension Library (IceLib) API.IceLib is a programming API that .NET developers can use to create custom applications that leverage the Customer Interaction Center. Installation and sample applications are also discussed.

For the latest version of this document, see the PureConnect Documentation Library at: http://help.genesys.com/pureconnect.

Table of Contents

Introduction to the IceLib API

The Interaction Center Extension Library (IceLib for short) is a programming API that allows developers to create custom applications that leverage CIC to solve business problems. IceLib is for developers who use modern .Net languages, such as C# or VB.Net.

IceLib provides a clean architecture that applications can use to manage sessions with CIC. IceLib provides session creation and login functions that allow applications to connect with one or more CIC servers using the login options that are available in the CIC clients (for example, user, password, station, remote number, remote station, persistent, and audio enabled). The figure below illustrates the tight coupling between IceLib and the Customer Interaction Center system architecture.
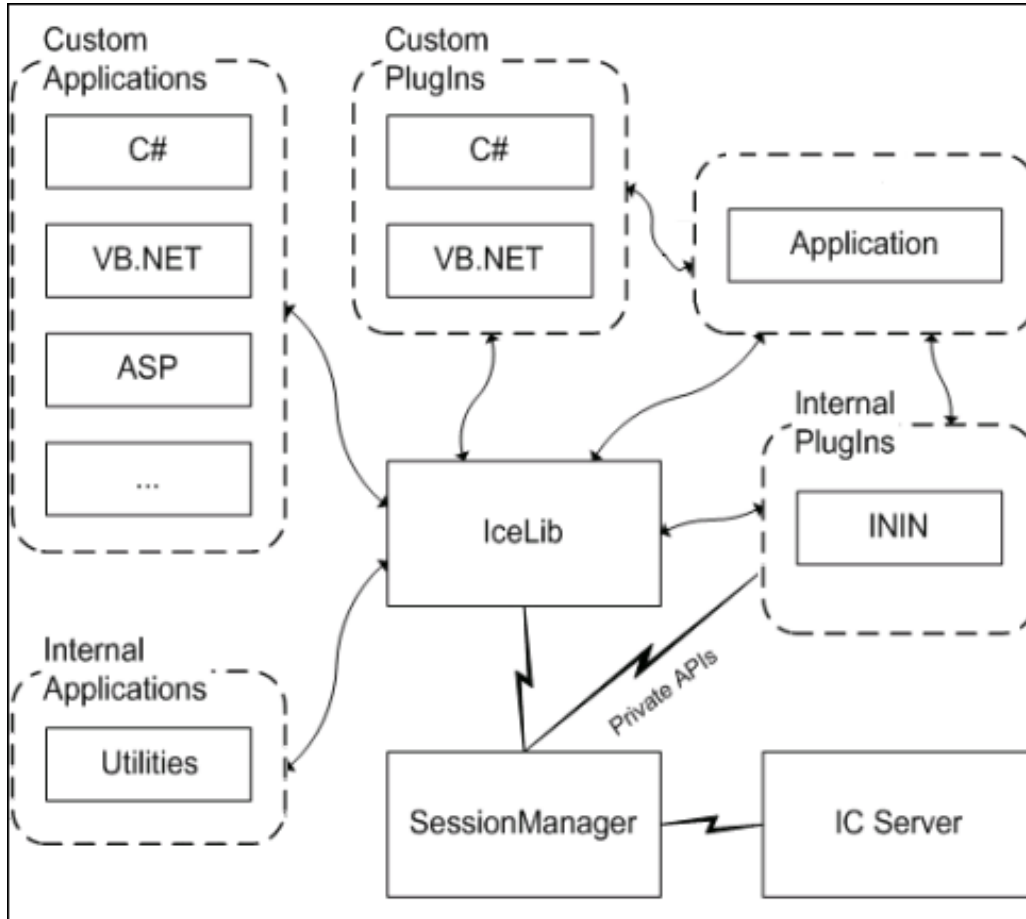
> **Note:**
> Because of underlying connection security changes, IceLib 3.0 integrations must be upgraded to use IceLib 4.0 in order to work with 4.0 or 201x Rx CIC servers. Conversely, IceLib integrations for CIC 4.0 or later cannot be used with 3.0 CIC servers.

IceLib 32-Bit and 64-Bit Support

The IceLib SDK has both a 32-bit and a 64-bit installation program, as well as a merge module for each. The two versions are necessary because IceLib .NET assemblies have dependencies on native 32-bit and 64-bit DLLs. For more information, see the IceLib API documentation topic "IceLib SDK 32-bit vs. 64-bit." IceLib 64-bit support is available inCIC 4.0 Service Update 4 and later releases.

Documentation

IceLib was professionally documented by its developers. The online documentation strongly resembles Microsoft's .NET documentation. In printable form, the IceLib documentation is over 12,000 pages. Moreover, IceLib assembly XML files provide *IntelliSense*© documentation when applications are developed in Visual Studio.

IceLib Technical Reference

Hide | Locate | Back | Forward | Stop | Refresh | Home | Print | Options

Contents | Index | Search

- Welcome to IceLib
- What's New in 4.0
- Getting Started
- Concepts
- ININ.IceLib Namespace
- ININ.IceLib.Configuration Namespace
- ININ.IceLib.Configuration.DataTypes Namespace
- ININ.IceLib.Configuration.EFaq Namespace
- ININ.IceLib.Configuration.Feedback Namespace
- ININ.IceLib.Configuration.Mailbox Namespace
- ININ.IceLib.Configuration.Mailbox.Utility Namespace
- ININ.IceLib.Configuration.OCS Namespace
- ININ.IceLib.Configuration.Optimizer Namespace
- ININ.IceLib.Configuration.ProcessAutomation Namespace
- ININ.IceLib.Configuration.Recorder Namespace
- ININ.IceLib.Configuration.Reporting Namespace
- ININ.IceLib.Configuration.Validators Namespace
- ININ.IceLib.Connection Namespace
- ININ.IceLib.Connection.Extensions Namespace
- ININ.IceLib.Data.TransactionBuilder Namespace
- ININ.IceLib.Directories Namespace
- ININ.IceLib.EFaq Namespace
- ININ.IceLib.Interactions Namespace
- ININ.IceLib.People Namespace
- ININ.IceLib.People.ResponseManagement Namespace
- ININ.IceLib.ProcessAutomation Namespace
- ININ.IceLib.QualityManagement Namespace
- ININ.IceLib.Reporting Namespace
- ININ.IceLib.Reporting.Interactions Namespace
- ININ.IceLib.Reporting.Interactions.Filters Namespace
- ININ.IceLib.Statistics Namespace
- ININ.IceLib.Statistics.Alerts Namespace
- ININ.IceLib.Tracker Namespace
- ININ.IceLib.UnifiedMessaging Namespace

Collapse All   Code: All

IceLib Technical Reference

ININ.IceLib Namespace

Send Feedback

The **ININ.IceLib** namespace contains fundamental classes and base classes for creating Interaction Center based applications. These include commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

Exceptions. There are a number of exceptions that can be thrown throughout IceLib. These are intended to allow custom applications to receive information about specific error conditions and be able to handle the condition accordingly. All such exceptions inherit from IceLibException. Examples of Interaction classes are: NotCachedException, ResourceInUseException, and ServerOperationNotSupportedException.

Tracing. The Tracing object offers support for simple tracing. These trace statements will be written into the Interactive Intelligence trace log for the running application instance. All such trace statements will be written to the "IceLib_Custom" trace topic.

Examples

The IceLib SDK includes example application references to the **ININ.IceLib** namespace. Some examples are listed below.

Project	Location
PeopleClient	WatchedUserStatusControl.cs, line 679, in WatchedUserStatusControl.SessionUserStatusChanged
	WatchedUserStatusControl.cs, line 754, in WatchedUserStatusControl.UserStatusListChangeWatchedCompleted
	WatchedUserStatusControl.cs, line 778, in WatchedUserStatusControl.UserStatusListStopWatchingCompleted
	UserDataQueryControl.cs, line 191, in UserDataQueryControl.UserDataSettingsStopWatchingCompleted
	UserDataQueryControl.cs, line 228, in UserDataQueryControl.UserDataSettingsStartWatchingCompleted
	UserRightsQueryControl.cs, line 359, in UserRightsQueryControl.UserRightsSettingsStartWatchingCompleted

ININ.IceLib.chm

IceLib Overview

IceLib interfaces with SessionManager, the CIC subsystem that brokers connections between client applications and a given CIC server. Custom IceLib applications fully leverage SessionManager, just like internally developed CIC applications. For example, Interaction Fax and Interaction VoiceMail both use IceLib and take advantage of its SessionManager capability. IceLib is feature-license based, not per-seat or per-session. Client sessions require client licenses, of course.

Generally speaking, IceLib provides the means to work with Interactions, Directories, People, Interaction Tracker, and Unified Messaging. It manages connections with the CIC server, specifies authentication and station settings, watches for connection state-change events, and performs actions relative to the connected Session user.

IceLib gives applications the ability to monitor *interactions* and *queues*. An interaction in a given queue can be watched for attribute changes. Applications can receive notifications when interactions are added or removed from a queue. Monitoring can be scoped to a given interaction, or to all interactions within a queue. Chats, Emails, and conferences can be monitored, along with telephone calls and other interaction types. In IceLib, objects and object watches contain only the data that the developer requests.

Change notification is implemented using events that follow the common Object/EventArgs pattern. This allows multiple notification recipients to be registered. It also allows recipients granular control over which notifications they receive. Developers should name the custom delegate for an event "FooEventHandler". Foo does not have to match the name of the event if FooEventHandler is generally useful for multiple events in the system. FooEventHandler's first parameter should be "object sender" and the second parameter should either be "EventArgs e" (and set to EventArgs.Empty) if no arguments are needed, or if a custom FooEventArgs class that inherits from EventArgs (or CancelEventArgs).

IceLib follows the direction that Microsoft has taken with public APIs. It conforms with Microsoft's design guidance and best practices for public APIs and framework development. It utilizes familiar style and naming conventions and it is based on Microsoft's .Net 2.0 technology framework.

For example, IceLib is object-based, rather than interface-based. This reflects the direction that Microsoft and the .NET technologies have taken to go beyond COM, in accordance with the .NET Framework Design Guidelines.

IceLib conforms with the Common Language Specification (CLS), a standard that defines naming restrictions, data types, and rules to which assemblies must conform if they are to be used across programming languages. This means that IceLib is compatible with C#, VB.Net, and all other .Net languages.

IceLib is a strongly typed API. Common attributes are accessible as properties, enumerations, or constants. IceLib supports generics, events, asynchronous patterns, and nullable types. Properties are used instead of public fields. This helps make the API future-proof, since changes can be made to a property get/set without affecting existing third-party application usage.

IceLib provides a consistent set of stable interfaces that expose Interaction Desktop feature sets to third-party applications. It provides a stable foundation for application development. IceLib's internal consistency promotes intuitive use, and its adherence with .Net eases understanding, reduces ramp-up time, and speeds development.

> **Note:**
> Customer Interaction Center (CIC) supports two interaction management client applications: Interaction Connect and Interaction Desktop.

This documentation uses the term CIC client to refer to either Interaction Connect or Interaction Desktop. For more information about CIC clients, see *CIC Client Comparison* in the PureConnect Documentation Library.

Consistent API design promotes intuitive use by application developers. Further, it can reduce the number of bugs when correct usage patterns have previously been learned. The ease of use improvements gained through design consistency are sometimes termed "The Power of Sameness". This is beneficial to customers since it can reduce application development costs.

IceLib provides synchronous (blocking) and asynchronous (non-blocking) versions of most methods, especially those methods that make a server call. This convention allows the developer the convenience of choosing which programming model to use. It also allows him to switch back and forth between the two models where appropriate within the same application.

In IceLib, errors are reported via exceptions rather than by returning or querying error codes. Methods are designed to *fail fast*, by evaluating parameters early and to throw meaningful argument exceptions. This helps developers to identify issues with code more quickly and easily.

IceLib ships with full-featured sample applications in several languages (C#, VB.NET, ASP.NET). Each application is commented and designed to illustrate "best practices" of a real implementation. The examples cover key topic areas, such as:
- Interactions, Queues, and Voicemail (C#)
- Workgroups, Users, and Statuses (VB.NET)
- User Rights, Access, Status Messages, Workgroups, etc. (C#)
- Directories and Statuses (ASP.NET)
- Directories Metadata & Paged Views (C#)
- More Directories (C#)

- Tracker Types Information (C#)
- Tracker Queries (C#)

How the IceLib API Help is Organized

The IceLib API help contains an introductory section about the ININ.IceLib namespace, which is the parent namespace for all the other namespaces, classes, and features of IceLib. The ININ.IceLib topic contains sub-topics for all the parent classes at the top level of IceLib. The introductory section also contains:

- A "What's New" page that describes the latest additions and improvements.

What's New in 4.0

Send Feedback

With the release of IceLib 4.0, there are a number of features that have been added, improved and changed to match new IC Server changes. Each namespace in IceLib is broken out to list each of these changes, to make it easier to find places where existing IceLib integrations may need to be updated for major breaking changes, or places where integrations could be enhanced to use newly introduced features. It is worth noting that, due to underlying connection security changes, IceLib 3.0 integrations must be upgraded to use IceLib 4.0 in order to work with 4.0 IC Servers. Conversely, IceLib 4.0 integrations cannot be used with 3.0 IC Servers.

The IceLib example apps have been upgraded to use the Visual Studio 2010 development environment, along with other improvements. The IceLib assemblies have been upgraded to use the .NET Framework 3.5. The IceLib assemblies were upgraded to use .NET 3.5 instead of .NET 4.0 so that systems utilizing IceLib integrations would not be required to upgrade to .NET 4.0; however, this does not restrict the ability to do so if desired.

Jump to a section:

- Documentation Improvements
- Example Applications

- A "Getting Started" section that describes some of the features of the IceLib API and the API Help.

Getting Started

Send Feedback

This section includes documentation on some key areas of the IceLib API, such as overview information and examples for common use cases. It can be challenging for a new developer to determine where to get started with the elements of a namespace. These topics are meant to "jump start" that effort, providing some important aspects to be aware of for a particular API area. Additionally, by providing focused example snippets around common tasks, these topics provide quicker understanding of what's possible and how to make those possibilities into reality.

✎ Tip

The quick start topics are a great place to get started with the elements of a namespace. Then, the IceLib SDK's more full-featured example applications can serve as a great resource for example code in the context of developing real applications that allow users to perform larger tasks..

Example Applications
 IceLib ships with full-featured sample applications in several languages (C#, VB.NET, ASP.NET).

Configuration

- A "Concepts" section that explains some of the basic ideas needed to understand and use IceLib.

Concepts

Send Feedback

This section includes documentation on some fundamental concepts which are common to many different areas of the IceLib API. Some concepts are simple, while others contain nuances that can trip up developers. These topics strive to clarify these general points, providing conceptual context for utilizing elements throughout the IceLib API.

Version Information
 IceLib API elements have documentation describing their release version information.

How Watches Work
 Many classes in IceLib require a watch to be active for certain properties to be accessible or for certain events to be raised.

How Async Calls Work
 IceLib provides synchronous (blocking) and asynchronous (non-blocking) versions of most methods, especially those methods that make a server call.

Integration Options

Structure of namespace sections

Each namespace topic consists of:
- An introductory section that gives an overview of the namespace, including its purpose and the general categories of classes that it contains. This section might also include additional background information. It might also include code examples to help you use the classes and other features more effectively.
- A list of links to sub-topics, each of which gives details about a class within the namespace.
- A list of links to sub-topics, each of which gives details about an enumeration within the namespace.
- A list of links to sub-topics, each of which gives details about an interface within the namespace.

Structure of class topics under namespaces

Each class topic consists of:
- An introductory section that gives an overview of the class, including its purpose.
- Definitions of the class's syntax in different languages, such as Visual Basic, C#, C++, J#, and JScript.
- Additional information needed to use the class effectively, including potential pitfalls.
- The inheritance hierarchy of the class.
- Information about thread safety with the class.
- A link to a topic that describes members of the class.

IceLib Namespaces

This API is encapsulated in a single root namespace (IceLib), located directly off of the ININ namespace. IceLib's 30 namespaces are arranged in a flat hierarchy that logically corresponds to high-level concepts and functionality. This arrangement simplifies discovery during development, makes IceLib easier for developers to use, and reflects the fact that IceLib is product-independent. The namespaces are:

ININ.IceLib

IceLib.Configuration

IceLib.Configuration.DataTypes

ININ.IceLib.Configuration.Efaq

ININ.IceLib.Configuration.Feedback

IceLib.Configuration.Mailbox

ININ.IceLib.Configuration.Mailbox.Utility

ININ.IceLib.Configuration.OCS

ININ.IceLib.Configuration.Optimizer

ININ.IceLib.Configuration.ProcessAutomation

ININ.IceLib.Configuration.Recorder

ININ.IceLib.Configuration.Reporting

IceLib.Configuration.Validators

ININ.IceLib.Connection

IceLib.Connection.Extensions

ININ.IceLib.Directories

IceLib.Efaq

ININ.IceLib.Interactions

ININ.IceLib.People

ININ.IceLib.People.ResponseManagement

ININ.IceLib.ProcessAutomation

IceLib.QualityManagement

ININ.IceLib.Reporting

ININ.IceLib.Reporting.Interactions

ININ.IceLib.Reporting.Interactions.Details

ININ.IceLib.Reporting.Interactions.Filters

ININ.IceLib.Statistics

ININ.IceLib.Statistics.Alerts

ININ.IceLib.Tracker

Tracker API

ININ.IceLib.UnifiedMessaging

ININ.IceLib

The ININ.IceLib namespace contains fundamental classes and base classes for creating Interaction Center based applications. These include commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

This namespace also facilitates exception handling. There are a number of exceptions that can be thrown throughout IceLib. These are intended to allow custom applications to receive information about specific error conditions and be able to handle the condition accordingly. All such exceptions inherit from ININ.IceLib.IceLibException.

The IceLib namespace also provides a Tracing class. The Tracing object supports simple tracing. Trace statements are written to the Interactive Intelligence trace log for the running application instance. All trace statements are written to the "IceLib_Custom" trace topic.

IceLib.Configuration

The ININ.IceLib.Configuration namespace contains classes for configuring a CIC server. It has rich support for the `RoleConfiguration`, `UserConfiguration` and `WorkgroupConfiguration` objects, as well as basic support for other objects such as `SiteConfiguration` and `WrapUpCodeConfiguration`.

There are a number of classes within the **ININ.IceLib.Configuration** namespace that provide support to the classes mentioned in the preceding summary. Examples of the supporting classes are enumerations, event argument classes, and delegates used by events within classes.

List-Based Configuration Objects

There are a number of object classes that are used to get properties for configuration objects that can have multiple instances on the server (distinguished by their `ConfigurationId`). These consist of `ListConfigurationObject`-derived classes. They are searched, with the results being cached, using a `ConfigurationList`-derived class. The search is composed of a QuerySettings instance which encapsulates a filter, a sort, the properties to be retrieved, the rights to be applied, and the result count limit. An example of this type of configuration object list is `UserConfigurationList` which is used to obtain `UserConfiguration` instances.

Some of these list-based configuration objects also support being edited, created, or deleted. These consist of `EditableListConfigurationObject`-derived classes and the `EditableConfigurationList`-derived classes used to search and cache them. The `EditableConfigurationList`-derived class has support for create and the `EditableListConfigurationObject`-derived class has support for edit and delete.

Container-Based Configuration Objects

There are a number of object classes that are used to get properties for configuration objects that can only have a single instance on the server. These consist of `ContainerConfigurationObject`-derived classes. They are queried, with the results being cached, using a `ConfigurationContainer`-derived class. The query is comprised of a which properties are to be retrieved.

Some of these container-based configuration objects also support being edited (but not created or deleted). These consist of `EditableContainerConfigurationObject`-derived classes. The `EditableListConfigurationObject`-derived class has support for edit. An example of this type of configuration object list is `SystemConfigurationContainer` which is used to obtain a `SystemConfiguration` instance.

IceLib.Configuration.DataTypes

The ININ.IceLib.Configuration.DataTypes namespace contains classes that provide support to the classes in the ININ.IceLib.Configuration namespace. Examples of the supporting classes are enumerations, event argument classes, and delegates used by events within classes.

ININ.IceLib.Configuration.Efaq

The `ININ.IceLib.Configuration.EFaq` namespace contains classes that provide support for EFaq configuration.

ININ.IceLib.Configuration.Feedback

The `ININ.IceLib.Configuration.Feedback` namespace contains classes that provide support for feedback configuration.

IceLib.Configuration.Mailbox

The ININ.IceLib.Configuration.Mailbox namespace contains classes that provide support to the classes in the ININ.IceLib.Configuration namespace. Examples of the supporting classes are enumerations, event argument classes, and delegates used by events within classes.

ININ.IceLib.Configuration.Mailbox.Utility

The ININ.IceLib.Configuration.Mailbox.Utility namespace contains classes to define, retrieve, set, and test IMAP servers, as well as to define and test various aspects of e-mail sending and delivery.

ININ.IceLib.Configuration.OCS

The `ININ.IceLib.Configuration.OCS` namespace contains classes that provide support for office communication server configuration.

ININ.IceLib.Configuration.Optimizer

The `ININ.IceLib.Configuration.Optimizer` namespace contains classes that provide support for optimizer configuration.

ININ.IceLib.Configuration.ProcessAutomation

The `ININ.IceLib.Configuration.ProcessAutomation` namespace contains classes that provide support for process automation configuration.

ININ.IceLib.Configuration.Recorder

The `ININ.IceLib.Configuration.Recorder` namespace contains classes that provide support for recorder configuration.

ININ.IceLib.Configuration.Reporting

The `ININ.IceLib.Configuration.Reporting` namespace contains classes that provide support for report configuration.

IceLib.Configuration.Validators

The ININ.IceLib.Configuration.Validators namespace contains classes that provide support to the classes in the ININ.IceLib.Configuration namespace. Examples of the supporting classes are enumerations, event argument classes, and delegates used by events within classes.

ININ.IceLib.Connection

The Connection namespace represents the functionality of creating a connection session with the server. Events are also sent for connection state changes and station settings changes. In addition, there is support to change the connected station and support to disconnect.

The following Authentication types are supported:
- Notifier (CIC username, password)
- Windows (username, NTLM (LAN manager) and/or Kerberos authentication)
- Restricted (used to obtain software updates)

The supported Station types are:
- Workstation (station name)
- Remote Station (station name, remote number, persistent)
- Remote Number (remote number, persistent)
- Stationless

The supported Host settings are:
- Host (BridgeHost machine name)
- Port (BridgeHost port)

The Connection namespace also provides reconnect functionality:
- Manually invoked reconnect attempts
- Optional auto-reconnect functionality
- Events that describe the state of auto-reconnect attempts (for user feedback).

IceLib.Connection.Extensions

The `ININ.IceLib.Connection.Extensions` namespace contains classes to support advanced connection-related features.

This includes retrieving ServerParameters, setting the user's password, persisting custom settings to the CIC server, sending or receiving a CustomNotification, and watching for creation of other Sessions.

ININ.IceLib.Data.TransactionBuilder

The ININ.IceLib.Data.TransactionBuilder namespace contains classes that client applications can use to execute transactions remotely through Transaction Server. This namespace contains supporting classes for executing transactions remotely through the xIC subsystem TransactionServer.

Application developers using the Tracker API will most likely not use the TransactionBuilder API directly. Classes and methods in the ININ.IceLib.Tracker namespace use classes in the ININ.IceLib.Data.TransactionBuilder namespace as a middle tier. This provides the communication layer that is used to execute remote stored procedures via TransactionServer.

The primary class used is the `TransactionClient` class. It represents the functionality of Transaction Builder, which is a programming interface that allows the execution of transactions using C# "code-genned" code.

The TransactionClient class:
- provides a low-level (middle tier) programming interface.
- allows execution of transactions using C# "code-genned" code.
- executes transactions remotely through xIC subsystem TransactionServer.

ININ.IceLib.Directories

The ININ.IceLib.Directories namespace contains classes for accessing Interaction Center contact directories. The namespace is organized around tasks one would typically want to perform, such as:

- Retrieve and manage the list of directories, and watch for changes to that list.

- Get available directories.
- Get directory metadata.

- Retrieving and managing entries within a specific directory.

- Get directory contacts.
- Get contact details.
- Perform paged directory views.
- Create, rename, and delete speed dial directories.
- Create, update, and delete contact entries.
- Link contact entries to a speed dial directory.

Contact directories contain lists of people and contact information about those people. For example, the default directory for private contacts is named IC Private Contacts, the default directory for public contacts is IC Public Contacts, and the directory which holds all company employees is the Company Directory. Classes in this namespace support every type of directory that CIC provides:

- Customer Directory
- Workgroup Directories
- Group and Profile Directories
- StationGroup Directories
- Speed Dial Directories
- General Directories implemented via the Data Manager

ININ.IceLib.Directories is composed of several application programming interfaces (APIs).

For more information, see the following:

- Directories Manager
- Basic Directory Support
- Watched View Support

Directories Manager

The Directories Manager is the key object to access all APIs for directory support. Its role is much the same as other managers found in IceLib such as the Interactions or People assemblies. That is to manage all resources internal to the assembly. The manager keeps track of who is watching what resources, maintains an interal cache of data retrieved and watches for change notifications to keep the watched objects up to date. In this assembly the Directories Manager will manage directory associated data and the watchers subscribing to that data.

All this functionality is opaque to the IceLib client customer. However, the Directories Manager is required to access all other APIs within this assembly for a given session.

Basic Directory Support

Classes in this API provide basic access to CIC directory data.

DirectoryConfiguration

This class gets a list of directories and their definitions (Directory Metadata) available. A directory definition, or metadata, includes it's name, owner, access permissions, category, and schema. The category is the type of directory: Company, Workgroup, Station Group, Speed Dial, Group and Profile and General Data Manager contact information. The schema provides a list of available columns and their definitions. Currently there are about 37 contact related columns supported.

DirectoryMetadata

This class represents the diretory metadata listed above.

ContactDirectory

A watched object that gets and caches all rows for a specific directory. Each row is represented by one or more Contact Entry objects. The Contact Directory will listen for notifications for any entries that have been changed, added or removed. The IceLib client application can subscribe to listen to these changes. The Contact Directory also provides the capability of performing client side caching of directory information, so that there is no impact on the network when a client session restarts.

ContactEntry

A directory entry represents one row within the directory cache. Where permitted directory entries can be modified, added or deleted. Currently in CIC this is mostly supported in the speed dial and general contact directories.

Watched View Support

The watched view provides a way to perform *paging* on contact data. Queries supply a start index, a count and an optional filter and sort order. These settings scope the query to only a handful of items. Change notifications are limited to items in the view.

ContactDirectoryWatchSettings: This class represents settings for a directory watch that provides a filtered and sorted view of contact entries. This view can be returned in increments or pages at a time.

This approach supports very large directories, since it limits the number of items that are returned at a time. The client application can then make requests to change the view by advancing next or going back a page at a time. A page is roughly defined as a *count* number of items. This is usually a small number, between 50 and 100. Setting this to a large number of entries would defeat the purpose of the paged approach.

IceLib.Efaq

The `ININ.IceLib.EFaq` namespace provides classes for accessing the e-FAQ Frequently Asked Question database.

To obtain a list of e-FAQ servers and their respective FAQ Topics, use the `EFaqServerList` class. This class provides access to the `EFaqServer` objects that correspond to the e-FAQ servers to which the user has been granted access permission. Queries can then be issued to each server via the Query method, or the corresponding asynchronous method, on the `EFaqServer` object.

ININ.IceLib.Interactions

The `ININ.IceLib.Interactions` namespace contains classes for manipulating Interaction Center interaction queues.

There are a number of classes within the `ININ.IceLib.Interactions` namespace that provide support to the classes mentioned in the preceding summary. Example of the supporting classes are enumerations, event argument classes, and delegates used by events within classes.

Queue Watches

There are a number of object classes that are used to get specific attributes for the Interactions contained in a given Interaction Queue. They are watched, meaning, that any time an attribute is changed, the internal cache of that object is kept up to date so anytime that attribute's property is referenced the current value will be returned. An event notification is available when any of the attributes changes in value.

> **Note:**
> If the entire contents of an InteractionQueue is to be watched, Queue watches are more efficient than having individual Interaction watches on each Interaction in the Queue.

Interaction Watches.

There are a number of object classes that are used to get specific attributes for a given Interaction. They are watched, meaning, that any time an attribute is changed, the internal cache of that object is kept up to date so anytime that attribute's property is referenced the current value will be returned. An event notification is available when any of the attributes changes in value. All such objects inherit from Interaction. Examples of Interaction classes are: `CallInteraction`, `ChatInteraction`, and `EmailInteraction`.

Interaction Conferences

The `InteractionConference` object provides information about a conference of interactions. Additionally, the `InteractionQueue` object provides support for watching attribute changes for Interactions within conferences contained on an interaction queue.

ININ.IceLib.People

The `ININ.IceLib.People` namespace contains classes for accessing Customer Interaction Center workgroups, status messages, users and users' statuses. It represents the functionality of Workgroups and Users.

The customer interacts with this feature by writing .Net code to the `ININ.IceLib.People` application programming interface (API). This API is made up of things an IceLib application writer may want to query or manipulate affecting a logged in agent. The capabilities include:

- Query Workgroups and their members.
- Query User access lists.
- Query User rights.
- Get available status messages.
- Get filtered status messages by user.
- Get and set User status.
- Workgroup activation.
- Get Response Management documents/items and add new user entries
- Get available Account Codes and WrapUp Codes.
- Get available Custom Buttons.
- Query and request licenses.

This namespace provides object classes that get specific settings for the session user. These objects are watched, meaning that any time an attribute is changed, the internal cache of that object is kept up to date. Whenever that attribute's property is referenced the current value will be returned from the cache. Applications can receive an event notification when any of the attributes changes in value.

Examples of watched attribute classes are:

- `UserRightsSettings`—the basic user rights settings for a CIC user.
- `UserAccessListsSettings`—the access control lists settings for a CIC user.
- `UserDataSettings`—miscellaneous data settings for a CIC user, such as the user's workgroup membership, supervisory rights, and greeting preference.
- `UserSettings`—basic settings for a CIC user and access to all watched attribute and object classes in the People namespace.
- `WorkgroupDetails`—the common details for a CIC workgroup, such as its members, queue type, activation status, and so forth.

Attributes become watched when the IceLib application writer queries for attributes of interest about a user. The library will then keep track of notifications, update the attribute and send an optional event when an attribute changes value.

This API can watch objects too. Watched objects are one or more objects that are of interest to the application. The library will query for the objects and maintain them in a cache. The library will watch for updates to these objects and will update these objects in the cache. An optional event is available when objects in this cache change or in some instances objects have been added or removed from the system.

Examples of watched object classes are:

- `Custom ButtonList`—list of custom buttons defined for a CIC user.
- `FilteredStatusMessageList`—list of status messages available for a CIC user.
- `StatusMessageList`—list of status messages defined in the CIC system.
- `UserWorkgroupActivationList`—list of workgroup activations for a CIC user.
- `UserStatusList`—the current status for a list of CIC users.

ININ.IceLib.People.ResponseManagement

Response Management is the general term for features that allow a user to send predefined responses, such as textual messages, URLs, or files to other persons participating in a Chat interaction. Messages, URLs, or files can be sent from your custom applications.

The `ININ.IceLib.People.ResponseManagement` namespace contains classes for retrieving Customer Interaction Center Response Documents and for editing user-specific Response Documents. These documents typically answer frequently asked questions. Agents can use the standard answers defined in Response Documents when he or she is participating in a chat session.

Response Management works as follows:

1. A web visitor requests an interactive Chat session.
2. An agent is alerted by Interaction Desktop (or a custom IceLib application). The agent *picks up* the interaction request. Interaction Desktop pops a dialog on the agent's workstation that allows the agent to begin an interactive typing session with the customer. The **Responses** tab of the agent's Chat dialog can contain the names of preset standard text messages, URLs which the agent can push the visitor's browser, and text file names. The agent can drag any combination of these responses into the **Response** field. When a URL is sent, the remote chat participant's web browser opens to that address.

   Agents can add personal responses, such as a personal greeting or a frequently sent file or URL, to the Response Management library. These personal responses are listed on the **Responses** tab under the *[User Name]*.xml directory, and are not available to other Interaction Desktop users.

Response Management Objects

A `ResponseItem` represents Interaction Messages, Interaction Urls and Interaction Files:

- An Interaction Message is a note, URL, or file that an agent can send to a web visitor during an interactive session. Interaction Message objects typically display short messages. For example, an Interaction Message titled 'Standard Response Times' could contain 'Standard response times for a support request are .....'.
- An Interaction Url stores frequently used Urls.
- An Interaction File points to a file path. For example, an Interaction File titled `IceLib Documentation` could point to `C:\ProgramFiles\InteractiveIntelligence\IceLib\Documentation\IceLib.chm`.

ResponseItemType

indicates if a `ResponseItem` is a Message, URL, File, or document.

ResponseDocument

is a collection of Interaction Messages, Interaction Urls and Interaction Files. A ResponseDocument may contain nodes that in turn contain a collection of Interaction Messages, Interaction Urls and Interaction Files. A response node can contain child nodes.

ResponseNode

is a collection of Interaction Messages, Interaction Urls and Interaction Files. A ResponseNode may contain more ResponseNodes and also ResponseItems.

ResponseManager

has the capability to retrieve Response Documents, Interaction Messages, Interaction Urls and Interaction Files. In addition, ResponseManager can also receive any updates at the server.

EditableResponseDocument

is a Response Document that can be edited by the application. By default, all documents are read-only. User can add response nodes and response items to the editable document. The editable can be obtained from the ResponseManager's UserDocument property.

EditableResponseItem

is a Response Item that can be edited by the application. By default, the ResponseItems are read-only. EditableResponseItems can be obtained from either EditableResponseNode or from EditableResponseDocument.

EditableResponseNode

is a Response Node that can be edited by the application. By default, the ResponseNodes are read-only. EditableResponseNodes can be obtained from either EditableResponseNode or from EditableResponseDocument.

ResponsesChangedEventArgs

indicates that there has been a change in Response documents at the server. Set a event handler for ResponsesChanged to receive updates.

ININ.IceLib.ProcessAutomation

The ININ.IceLib.ProcessAutomation namespace contains classes for manipulating, monitoring, and managing Interaction Process Automation data.

The Interaction Process Automation system consists of many entities such as Processes, Process instances, and Work Items.

IceLib.QualityManagement

The ININ.IceLib.QualityManagement namespace contains classes for implementing quality management related features using the IceLib API and Interaction Center.

ININ.IceLib.Reporting

The `ININ.IceLib.Reporting` namespace contains classes for implementing reporting related features using the IceLib API and Interaction Center.

`InteractionSnapshotBrowser` can be used to retrieve historical information about interactions. Use `GetInteractionSnapshot(GetInteractionSnapshotParameters)` to retrieve historical data about a specific interaction, or use `GetInteractionSnapshots` to retrieve a filtered list of historical data about multiple interactions.

ININ.IceLib.Reporting.Interactions

The `ININ.IceLib.Reporting.Interactions` namespace contains classes that provide support for Interactions reporting classes.

ININ.IceLib.Reporting.Interactions.Filters

The `ININ.IceLib.Reporting.Interactions.Filters` namespace contains classes that provide support for filtering interaction snapshots via GetInteractionSnapshots.

ININ.IceLib.Reporting.Interactions.Details

The `ININ.IceLib.Reporting.Interactions.Details` namespace contains classes that examine the segments of an interaction. This includes information such as the workgroup and parties involved with the segment, as well as basic information such as segment type, duration, and disposition.

ININ.IceLib.Statistics

The `ININ.IceLib.Statistics` namespace contains classes for watching and listening to Interaction Center statistics.

There are a number of classes within the `ININ.IceLib.Statistics` namespace that provide support to the classes mentioned in the preceding summary. Examples of the supporting classes are enumerations, event argument classes, and delegates used by events within classes.

The `StatisticCatalog` can be used to watch statistics. This watch will provide the details about the definition of the statistic, such as units, precision, name, and description. The `StatisticListener` can be used to listen to statistic values. This will retrieve the current value of the statistic of interest and send out events when that value changes.

ININ.IceLib.Statistics.Alerts

The `ININ.IceLib.Statistics.Alerts` namespace contains classes for viewing, changing, and receiving Interaction Center statistics alerts.

There are a number of classes within the `ININ.IceLib.Statistics.Alerts` namespace that provide support to the classes mentioned in the preceding summary. Examples of the supporting classes are enumerations, event argument classes, and delegates used by events within classes.

The following sequence describes one way of utilizing the members of the `ININ.IceLib.Statistics.Alerts` namespace for creating an alert to monitor.

- Use `StatisticCatalog` to get a list of the available `StatisticDefinition`s
- Use a `StatisticDefinition` to create a `StatisticKey`, which would describe the threshold for an alert
- Use the `StatisticKey` to create an `AlertDefinition`
- Create `AlertRule`s associated with the `AlertDefinition`
- Add the `AlertDefinition` to an `EditableAlertSet`
- Create the `AlertSet` by sending the `EditableAlertSet` to the `CreateAlertSet(EditableAlertSet)` method
- Subscribe to notifications for the `AlertSet` by calling the `Subscribe(AlertSet, Boolean)` method on `AlertCatalog`

Memo Watches

A `Memo` is an object that can be sent out from the server under certain circumstances or sent from a user to a list of recipients. The `MemoList` is used to watch for `Memo` objects that are sent out. There are three types of watching that can take place, MyMemos, GeneralMemos, and AlertMemos. MyMemos watch for `Memo` objects that target the current user. GeneralMemos watch for all non-alert `Memo` objects. AlertMemos watch for any `Memo` objects that are associated with an `AlertSet`. The `MemoList` is also used for sending out new `Memo` objects and sending out `Memo` updates.

ININ.IceLib.Tracker

The ININ.IceLib.Tracker namespace contains classes for manipulating and managing Interaction Tracker data. The Interaction Tracker system consists of many entities such as Interactions, Organizations, Individuals, AddressTypes, and InteractionAddressTypes. Together, these entities provide Interaction Center users with a robust means of managing contact information, tracking interactions, and retrieving information pertaining to these different entities from the Tracker database.

About the Tracker xIC Subsystem

If you are unfamiliar with Interaction Tracker, it is a subsystem that runs on the xIC server. It's job is to process events and notifications from the other xIC subsystems. In doing so, the Tracker subsystem updates the Tracker database based on these Tracker related events (i.e. an interaction has been completed, a participant has been added to or removed from a segment, etc.).

Tracker API

The Tracker API consists of a set of classes that allows a .NET developer to create an application that accesses Tracker data. Since the data access is provided through the IceLib Tracker API, the developer does not need to write any low-level database code. A very robust application may be written in any .NET language without the need for any ADO.NET code.

> **Note:**
> Any application that uses the Tracker API requires a valid Session Manager session. The logged in user of the Session Manager session must have a Tracker Access license, otherwise, the Tracker methods will not be executed.

The .NET *Nullable Types* feature is a good fit for some aspects of IceLib. In particular, the Tracker API benefits, since it contains examples of *query by example*, where the properties of an object should only be matched if they are specified (i.e. not null).

There are two ways to declare nullable types: as a generic (e.g. `Nullable<int> foo;`) or as a type shortcut (e.g. `int? foo;`). Either of these are acceptable in the public API, although using the generic declaration might be more eye catching in the code. The documentation system always uses the generic approach, regardless of which way it is declared in the code.

Nullable types can only be declared for value types; if a type is already a reference type (even an immutable one like `string`), it cannot be further wrapped up into a nullable type. Therefore `Nullable<string>` will not compile. Since reference type variables can already hold null values, this isn't really a problem.

You can use the Tracker API to:

- Administer tracker metadata types.
- Perform Tracker user operations.
- Search Tracker information.

The three main classes in the ININ.IceLib.Tracker namespace are:

1. `TrackerAdmin`. This class provides methods that allow you to add, delete, update and retrieve Tracker types, and perform administrative tasks. The Tracker types are:
   - AddressType - used to classify addresses within Tracker.
   - InteractionAddressType - used to classify interaction addresses within Tracker.
   - InteractionAddressSubtype - used to further classify interaction addresses within Tracker.
   - IndividualType - used to classify Individuals within Tracker.
   - OrganizationType - used to classify Organizations within Tracker.
   - TrackerAttributeType - used to classify Attributes within Tracker.
   - Title - represent titles that can be assigned to individuals (Mr., Miss, Dr., etc.)

   You will be familiar with these types if you have ever performed a Tracker administrative task in Interaction Administrator. The Tracker node in Interaction Administrator allows administrators to add, delete, and update AddressTypes, IndividualTypes, OrganizationTypes, etc. If you want to create a .Net application that provides administrative capabilities for Tracker, you would use the TrackerAdmin class.

2. `TrackerUser`. This class provides methods that allow you to add, delete, update and retrieve Tracker data that relates to Individuals, Organizations, and Locations. These entities are represented by minor classes in the ININ.IceLib.Tracker namespace:
   - Individual - represents an Individual within Tracker.
   - Organization - represents an Organization within Tracker.
   - Location - represents a Location within Tracker.
   - Annotation - represents an Annotation within Tracker.
   - IndividualAddress - represents an address for an Individual within Tracker.
   - IndividualInteractionAddress - represents an interaction address for an Individual within Tracker.
   - OrganizationAddress - represents an address for an Organization within Tracker.
   - OrganizationInteractionAddress - represents an interaction address for an Organization within Tracker.
   - LocationAddress - represents an address for a Location within Tracker.
   - LocationInteractionAddress - represents an interaction address for a Location within Tracker.

   You will be familiar with these Tracker entities if you have ever used the Win32 TrackerClient application. The TrackerClient allows you to manage Tracker Contacts. You can add an address for an Individual. You can associate an Individual with a particular Organization. You can add a new Organization or Location. If you want to create a .Net application that provides these user level capabilities for Tracker, you would use the TrackerUser class.

3. `TrackerSearch` contains methods that perform search operations. These methods allow you to search the Tracker database for Individuals, Organizations, Locations, and Interactions. The classes used by TrackerSearch are:

- IndividualView - contains information about an Individual within Tracker.
- Organization - contains information about an Organization within Tracker.
- LocationView - contains information about a Location within Tracker.
- InteractionView - contains information about an Interaction within Tracker.

ININ.IceLib.UnifiedMessaging

This namespace manages *Faxes* and *Voicemails*. Unified Messaging classes allow access and manipulation of Fax and voicemail files generated by a CIC server. Typically these files are delivered to a mailbox through the corporate Microsoft Exchange or IBM Notes server, although PureConnect also supports the use of file-based messaging in cases where Exchange or Notes is not in use.

Unified Messaging provides the means to:
- Manage and playback voicemails to handset, number, or station.
- Receive events for new voicemails.
- Manipulate and submit Faxes to the CIC server.
- Monitor outgoing Fax activity.
- Control MWI status and events.

The API provides both synchronous (blocking) and asynchronous (non-blocking) versions of each method, so that developers can choose the programming model that is most appropriate for their needs.

Fax Capabilities

Fax-related classes allow a .NET developer to manipulate and submit Faxes to a CIC server for sending, as well as monitor all outgoing fax activity for a logged-in user. If the need arises, the ability to cancel a pending Fax is also available.

Building a Fax

Interaction Faxes are separated into a collection of fax pages, page attributes and envelopes all available through a central object, the `FaxFile`.

Fax methods in the Unified Messaging API allow developers to create a new Fax file. The `FaxFile` object provides all of the functions necessary for the construction of a fax, including the ability to add and insert pages as well as save the file in TIFF or i3f format.

Sending a Fax

The Unified Messaging API can send a Fax through the CIC server. Faxes are addressed through the use of Fax envelopes. A single fax file contains one envelope per addressee. Envelopes must be added before the Fax file can be submitted to the CIC server for delivery.

Monitoring Fax Activity

The Unified Messaging API can monitor outgoing Fax activity. When monitoring is enabled, an application will receive periodic update events about the state of each Fax being processed on the CIC server.

Voicemail Capabilities

Voicemail functionality is implemented by classes that allow a .NET developer to playback voice messages received by a CIC server as well as to monitor for new incoming messages. Developers can tap into the voicemail recording capabilities of the CIC server in order to create new voicemail audio.

Playback functionality is provided through the `VoicemailMessage` object. It allows a developer to play a voice message to any of the following locations:
- Handset - the handset associated with the station a user is currently logged into. In the case of SIP audio, the handset is a set of speakers or headphones. In a traditional phone setup, the handset is typically a phone receiver.
- Number - a remote phone number
- Station - a station defined in CIC

The API also offers the ability to control the state of a voicemail by marking it as read (acknowledged) or not.

File Access Capability

The API can download the voicemail WAV file to a local workstation. The access capabilities include retrieval of existing voicemail audio files attached to a message, and creation and retrieval of new voicemail audio files.

MWI Capability

The final piece of functionality in the unified messaging API allows a developer to control the state of their voicemail waiting indicator. The indicator can be manually set to on or off or the server can be instructed to calculate the correct state through the API.

Install IceLib

To install IceLib (32-bit) or (64-bit) on a Windows computer:

1. If you have not done so already, follow the procedure on the PureConnect Product Information site at https://my.inin.com/products/cic/Pages/Releases-and-Patches.aspx to download and copy the CIC 2015 R1 or later `.iso` file to a file server on the CIC network.
2. Run `Install.exe` from the `\Installs` directory on the share.
3. Locate the IceLib (32-bit) or (64-bit) install in the **Off-Server Components** tab, click the checkbox, and click **Install**.
4. Click **Next** in the first and second dialogs.
5. Click **Install** in the final dialog.

Customers can use a separate `msi` install in their own IceLib deployment scenarios. The Developer install copies the IceLib assemblies (and dependencies) onto the destination machine. These could be used as a sub-install of the IceLib Developer Install, to copy only the runtime assemblies necessary to use IceLib from a custom application. This offers simplicity and can be run from other installs, such as a customer's own `msi` projects.

Applying new releases

For instructions on applying new releases for IceLib, see https://my.inin.com/products/cic/Pages/Latest-Release.aspx.

Sample Applications

The IceLib Developer Install copies sample applications to an `ExampleApps` folder below the destination path. These practical working examples illustrate "best practices" use of IceLib. In total, they exercise the majority of IceLib's functionality.

| Applications in ExampleApps Folder		
Language	**Project Folder**	**Description**
C#	DirectoriesClient	Demonstrates the functionality of the ININ.IceLib.Directories namespace. This application shows how to connect to a CIC server, display information about contact directories on the server, view metadata for a directory, display entries in a directory, and even create a new contact directory.
C#	DirExplorer	C# source in the ExampleApps\DirExplorer folder exercises the functionality of the ININ.IceLib.Directories namespace by displaying workgroups, contact directories, Tracker, Station groups and entries in the Company Directory. A column in each view indicates whether or not a watch is in effect.
C#	FaxSample	This application demonstrates how to send a Fax message.
C#	InteractionsTest	This C# example implements a simple but feature-rich telephony client, similar to Interaction Desktop. You can perform common telephony tasks (place calls, send calls to voicemail, disconnect, etc.), view voicemails and query server parameters. This example exercises the functionality of the ININ.IceLib.Interactions namespace.
C#	ConnectionExtensionsTest	This application tests server parameters, password policies, and custom notifications within the IceLib API.
C#	StatisticsSample	This example exercises the functionality of the ININ.IceLib.Statistics namespace. It contains browsing of the statistics catalog, watching valid parameter values, listening to statistic value changes, and configuration and monitoring of alerts.
C#	PeopleClient	This example exercises the functionality of the ININ.IceLib.People namespace. Once you establish a session with the server, you can view **Watched Queries** (attributes of various settings, details of access rights, etc.), information about Status Messages, and license information.
C#	TrackerAdmin	This example helps you understand classes in the ININ.IceLib.Tracker namespace. The TreeView in the left pane allows you to select Address Types, Attribute Types, InteractionAddress Types, InteractionAddress Subtypes, Titles, Individual Types, and Organization Types. These elements correspond to classes in the Tracker namespace.
C#	TrackerClient	This example demonstrates the search functionality of the ININ.IceLib.Tracker namespace. It allows the user to search the Tracker for individuals, interactions, organizations, or a location.
VB.Net	VBTest	Visual Basic.Net source in the ExampleApps\VBTest folder demonstrates functionality of the ININ.IceLib.People namespace, especially the UserStatus and UserStatusList classes. Once you establish a session with the CIC server, you can filter workgroups to select persons in a given status.
ASP.Net	AspIceLibClientDemo	This example implements a web page to set status information and place a call. It shows how to use the IceLib public API to do tasks commonly carried out by CIC Client applications.
C#	ConfigurationCmdlet	This example shows one way to use the IceLib SDK to configure a CIC Server. It is a set of Windows PowerShell Cmdlets which are registered through the use of the SnapIn.
C#	AutomationProcessLauncher	This shows some capabilities of the ININ.IceLib.ProcessAutomation namespace. This is a simple console-based application that demonstrates launching an Interaction Process Automation process.

DirectoriesClient C# Example

C# source in the `ExampleApps\DirectoriesClient` folder demonstrates the functionality of the ININ.IceLib.Directories namespace. This application shows how to connect to a CIC server, display information about contact directories on the server, view metadata for a directory, display entries in a directory, and even create a new contact directory.
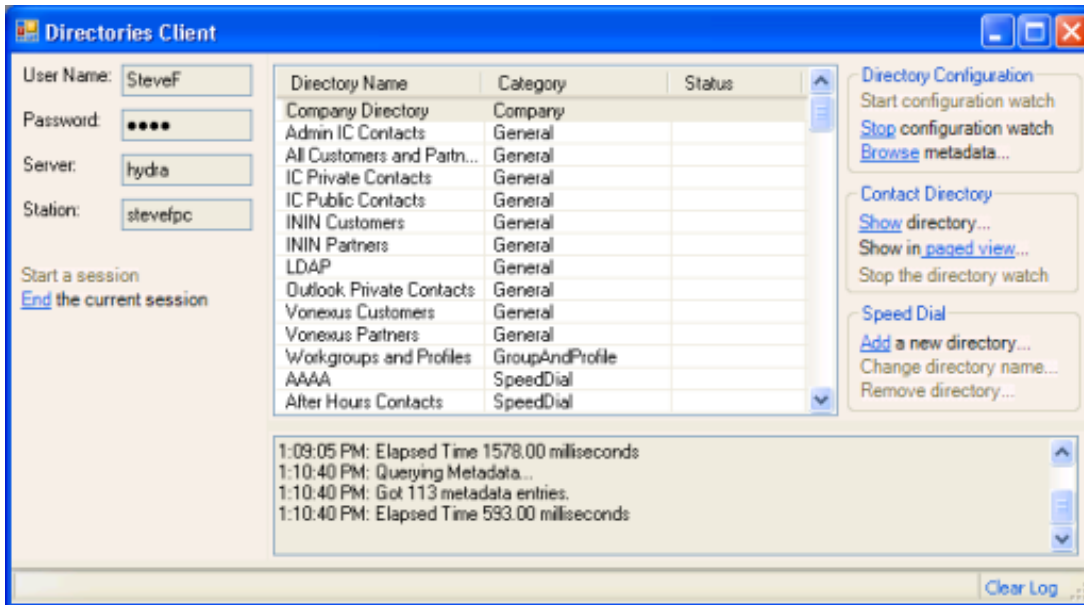
At startup, the application prompts for information needed to establish a session. Once a session is established with the server, a list of contact directories is displayed.
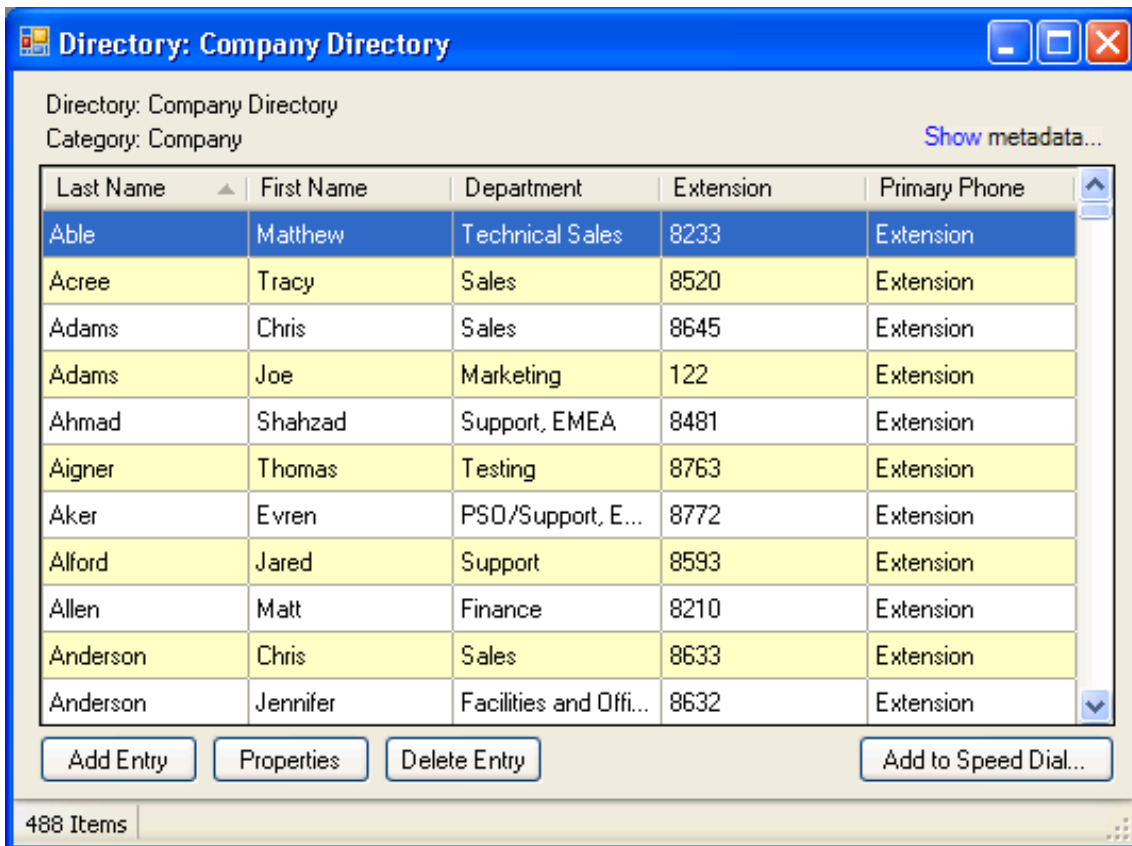
This dialog displays a *page view* of contact directory entries. You can add, remove, and display the properties of individual entries, or add entries to a speed dial directory on the server.

DirExplorer C# Example

C# source in the `ExampleApps\DirExplorer` folder exercises the functionality of the ININ.IceLib.Directories namespace by displaying workgroups, contact directories, Tracker, Station groups and entries in the Company Directory. A column in each view indicates whether or not a watch is in effect.
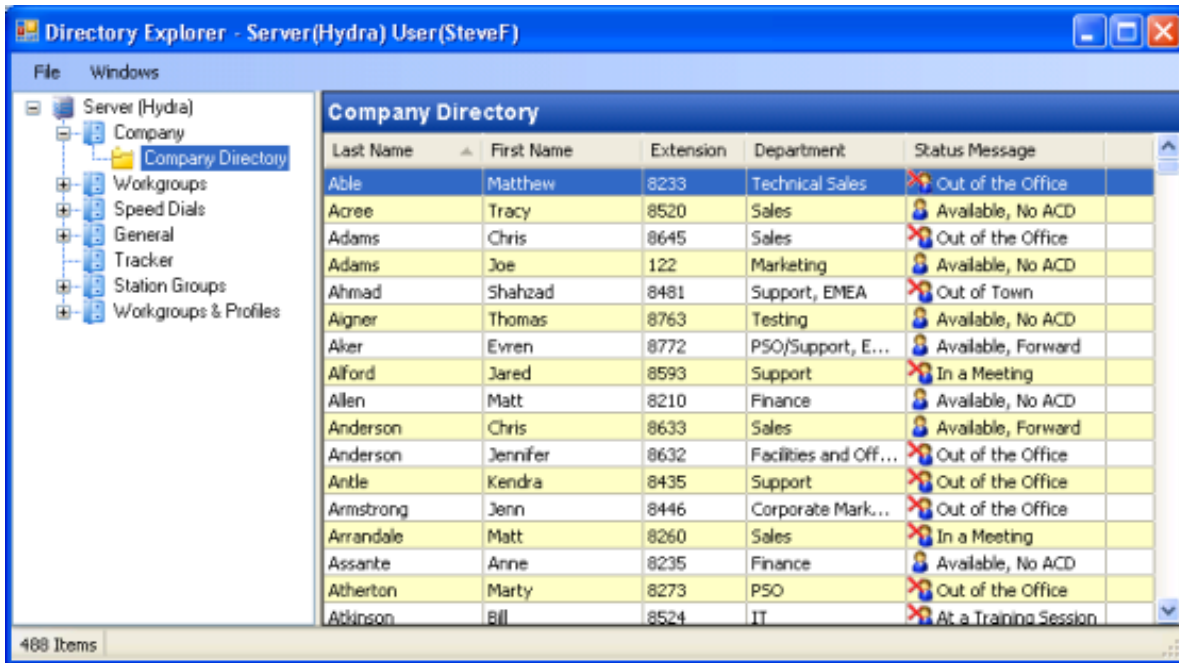
This application uses a simple two-pane window. The right pane displays details for the item selected in the ListView control on the left.

FaxSample C# Example

FaxSample is a C# application that demonstrates how to send a Fax via CIC.

1. Open `C:\ProgramFiles\InteractiveIntelligence\IceLibSDK\ExampleApps\FaxSample\FaxSample.sln`.
2. Press `Shift` + `F6` to build FaxSample. Then press `F5` to run it. The **Fax Sample** window will appear:
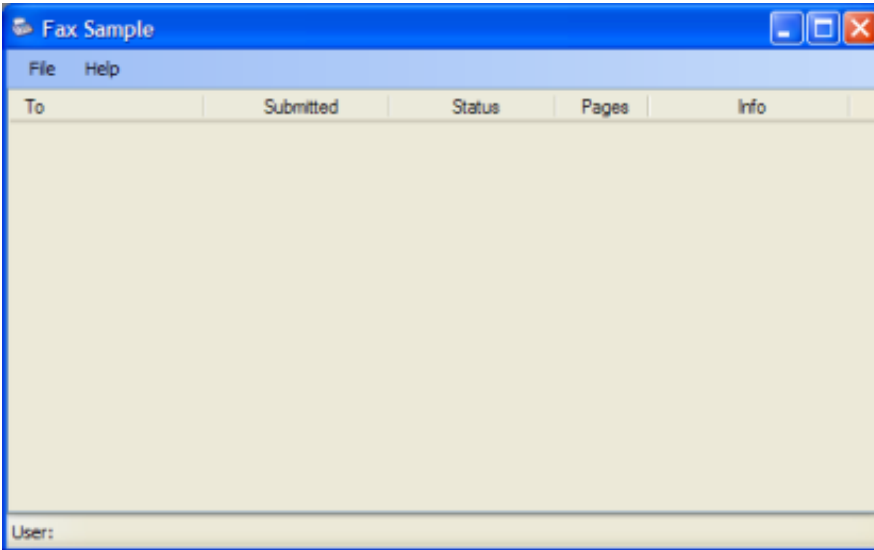
3. Pull down the **File** menu and select **Connect**. The **Send Test Fax** dialog will appear:
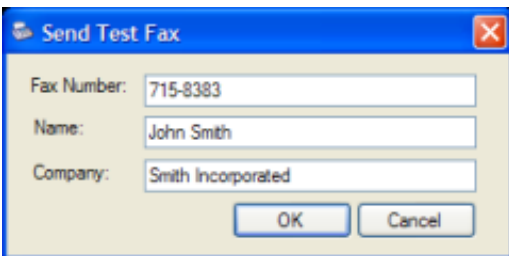
4. Enter destination Fax number, name, and company information. Then click **OK**.
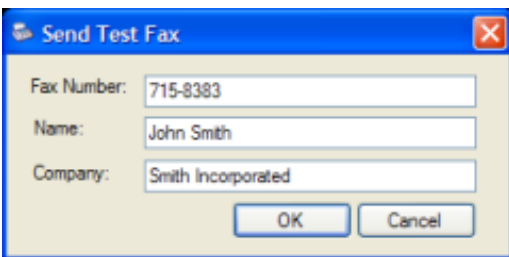
5. Wait for Fax transmission to end.
6. Pull down the **File** menu and select **Disconnect**.
7. When you are finished, pull down the **File** menu and select **Exit**.

InteractionsTest C# Example

This C# example implements a simple but feature-rich telephony client, similar to Interaction Desktop. It exercises the functionality of the ININ.IceLib.Interactions namespace. You can perform common telephony tasks (place calls, send calls to voicemail, disconnect, etc.), view voicemails and query server parameters. This source code is in the `ExampleApps\InteractionsTest` folder.
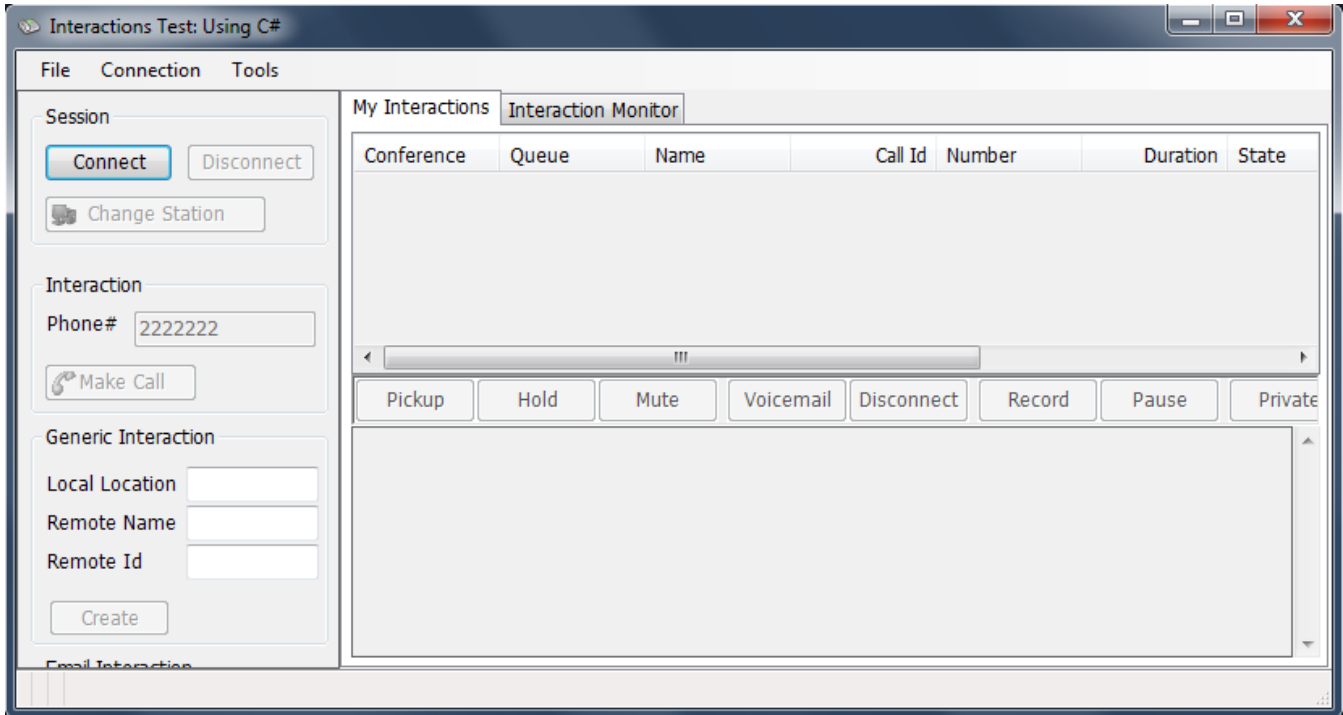
This application displays interactions in the logged in user's queue, and allows you to perform operations on them.

ConnectionExtensionsTest C# Example

This C# application tests server parameters, password policies, and custom notifications within the IceLib API.

Connection Extensions Test

Custom Messages to Handlers that can send a response

Object Id: [   ]

Event Id: [CustomCallDisconnect]

Request: [   ]   [Send]

Response: [   ]

Custom Messages between Icelib Sessions

Message Types (Enter in the format: Object Id,Event Id; ObjectId,EventId;)

[   ]   [Start Watch]
[   ] [Change Watch]
[   ] [Stop Watch]

Last Message Received

[   ]

Edit your Response

[   ]   [Send]

Custom Messages to Handlers that DO NOT send a response

Object Id: [   ]

Event Id: [   ]

Request: [   ]   [Send]

Custom Messages to between Icelib Sessions

Object Id: [   ]

Event Id: [   ]

Request: [   ]   [Send]

Response: [   ]

Connection Extensions Test

Establish Connection | Server Parameters | **Password Policies** | Custom Notifications

☑ Show Password Policy

Change password

Current Age: -1
Maximum Age: 180
Maximum Nag Period: 14
Minimum Age: 2
Minimum Length: 4
Uniqueness Constraint: False
Al Sequential Digits Allowe: 12/31/1969 7:00:00 PM
Last Password Set Time: 5
Maximum Allowed Login Attempts: 30
AccountLockoutDuration: 30
AccountLockoutResetTime: 12/31/1969 7:00:00 PM
LastLockoutTime: False
Must Change Password: False
Must Change Password Time: 12/31/1969 7:00:00 PM

Old Password   [   ]

New Password   [   ]

[Change]   [Cancel]

Connection Extensions Test

Establish Connection | Server Parameters | Password Policies | Custom Notifications

Enter the Server Parameters separated by comma

[e-FAQ Support,Vendor Support,Dialer Support]   [Add]   [Stop Watch]

Name	Exists	Value
Dialer Support	No	
e-FAQ Support	Yes	0
Vendor Support	Yes	dialogic

[Remove]

Statistics C# Example

This example exercises the functionality of the ININ.IceLib.Statistics namespace. It contains browsing of the statistics catalog, watching valid parameter values, listening to statistic value changes, and configuration and monitoring of alerts.
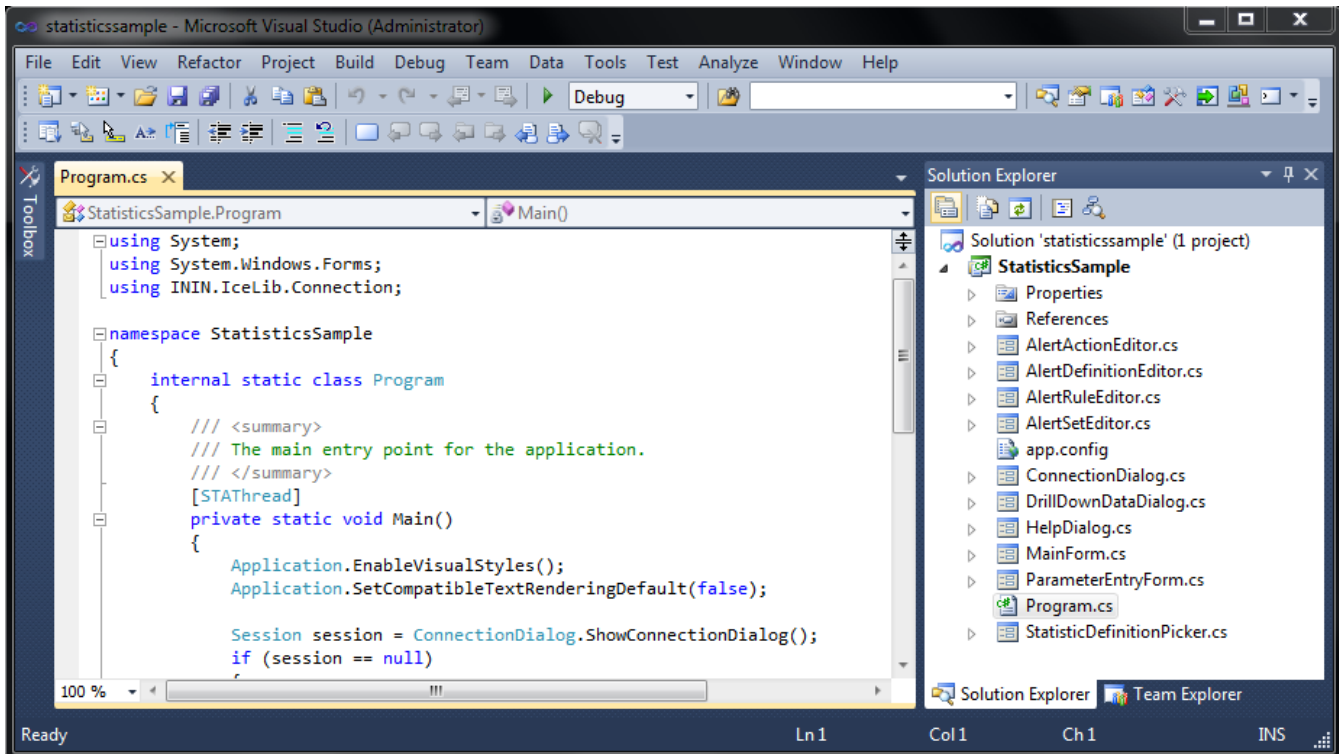
PeopleClient C# Example

This example exercises the functionality of the ININ.IceLib.People namespace. Once you establish a session with the server, you can view **Watched Queries** (attributes of various settings, details of access rights, etc.), information about **Status Messages**, and **license information**.
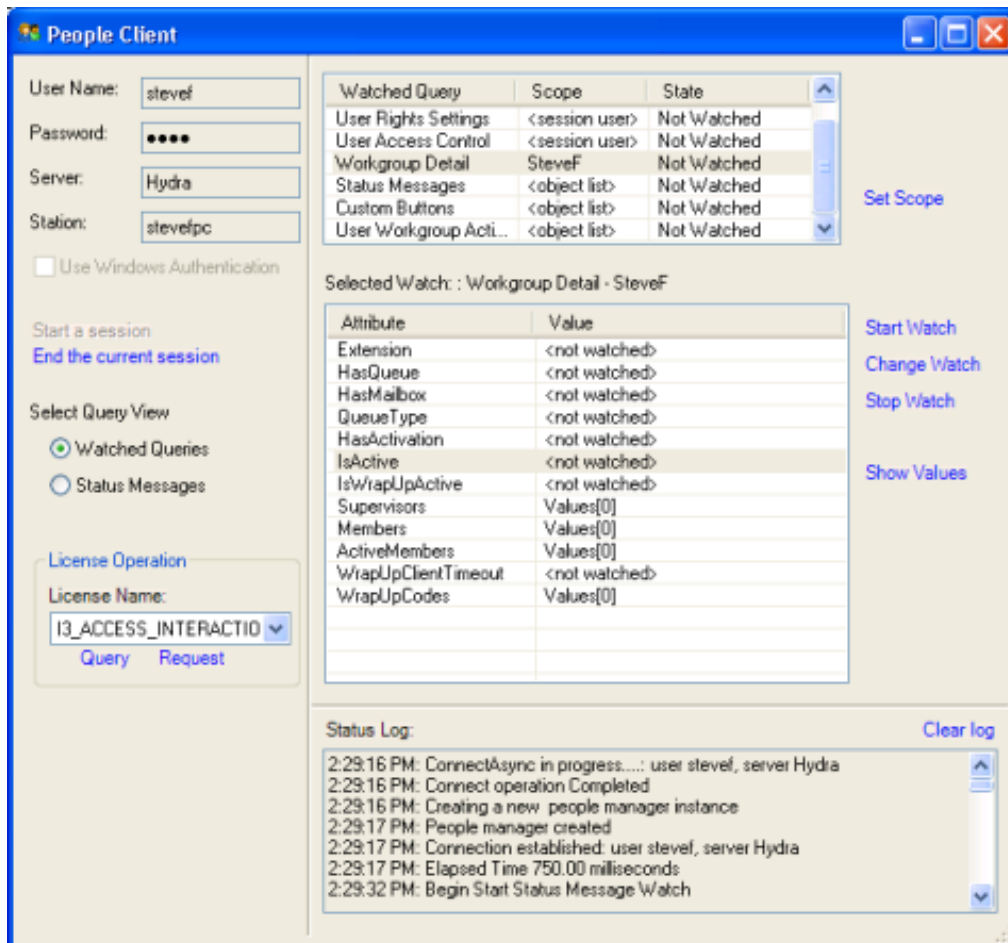
Watched queries display the attributes of user data settings, user rights, access control, workgroup details, and so forth.
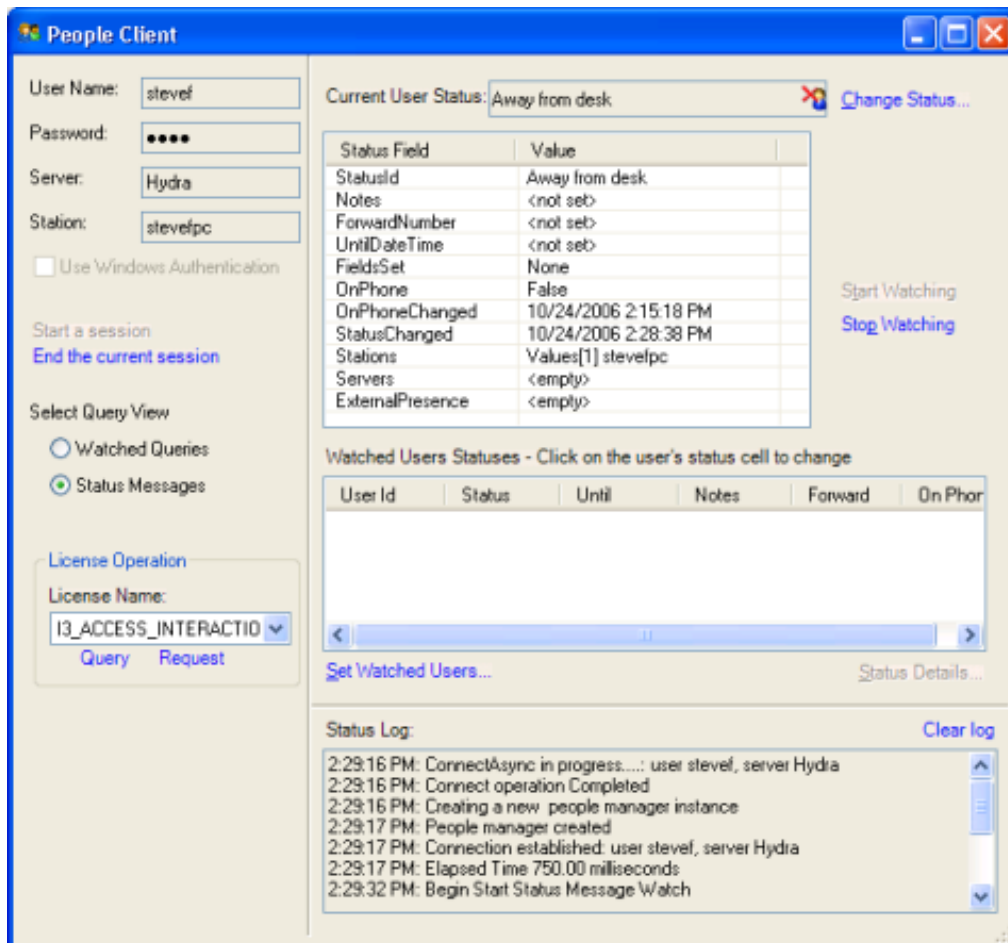
When you set the Query View to **Status Messages**, you can monitor and edit the status of the logged in user, set watches on the status of other users, and more.

TrackerAdmin C# Example

C# source in the `ExampleApps\TrackerAdmin` folder helps you understand classes in the ININ.IceLib.Tracker namespace. The TreeView in the left pane allows you to select Address Types, Attribute Types, InteractionAddress Types, InteractionAddress Subtypes, Titles, Individual Types, and Organization Types. These elements correspond to classes in the Tracker namespace.

For example, *AddressTypes* classify addresses within Tracker. The Name property of an AddressType is what is often displayed in applications. Typical AddressType names are: billing, business, home, and shipping.

OrganizationTypes classify Organizations within Tracker. Typical OrganizationType names are: System, External, Internal, Customer, Partner, Vendor, and so on.

InteractionAddressTypes classify interaction addresses within Tracker. Typical InteractionAddressType names are: **Chat**, **Email**, **Fax**, and **Pager**.

Each IndividualInteractionAddress, LocationInteractionAddress, and OrganizationInteractionAddress contains an InteractionAddressType member that is used to hold the address type that it represents. A typical **IndividualInteractionAddress** may contain an interaction address type of **Phone** and an interaction address subtype of **Assistant**.
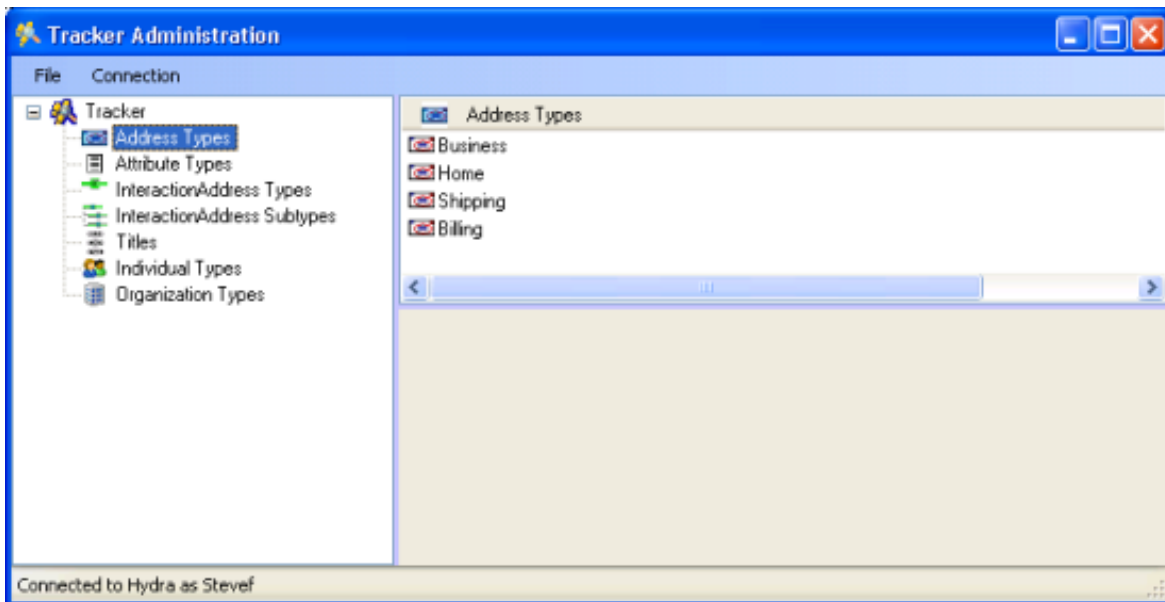
AddressTypes in Interaction Tracker.

TrackerClient C# Example

C# source in the `ExampleApps\TrackerClient` folder demonstrates the functionality of the ININ.IceLib.Tracker namespace. This application searches the Tracker database for individuals, interactions, organizations, or location.
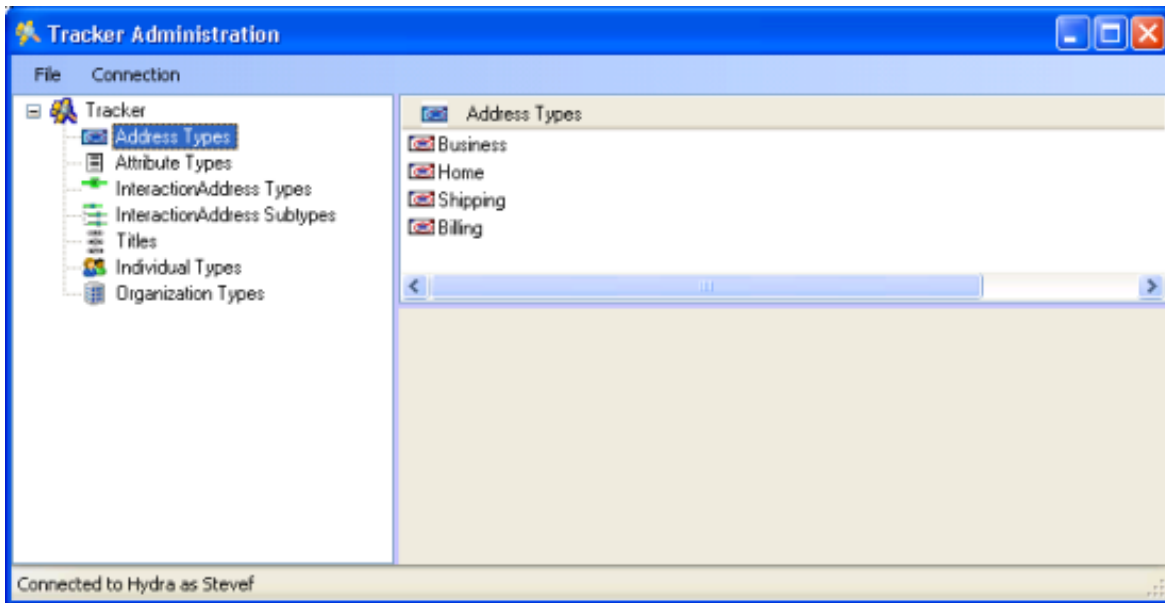
Example location search returns records for Fishers, Indiana.

VBTest VB.NET Example

Visual Basic.Net source in the `ExampleApps\VBTest` folder demonstrates functionality found in the ININ.IceLib.People namespace, especially the UserStatus and UserStatusList classes. Once you establish a session with the CIC server, you can filter workgroups to select persons in a given status. For example, you can quickly display a list of people who are on vacation.
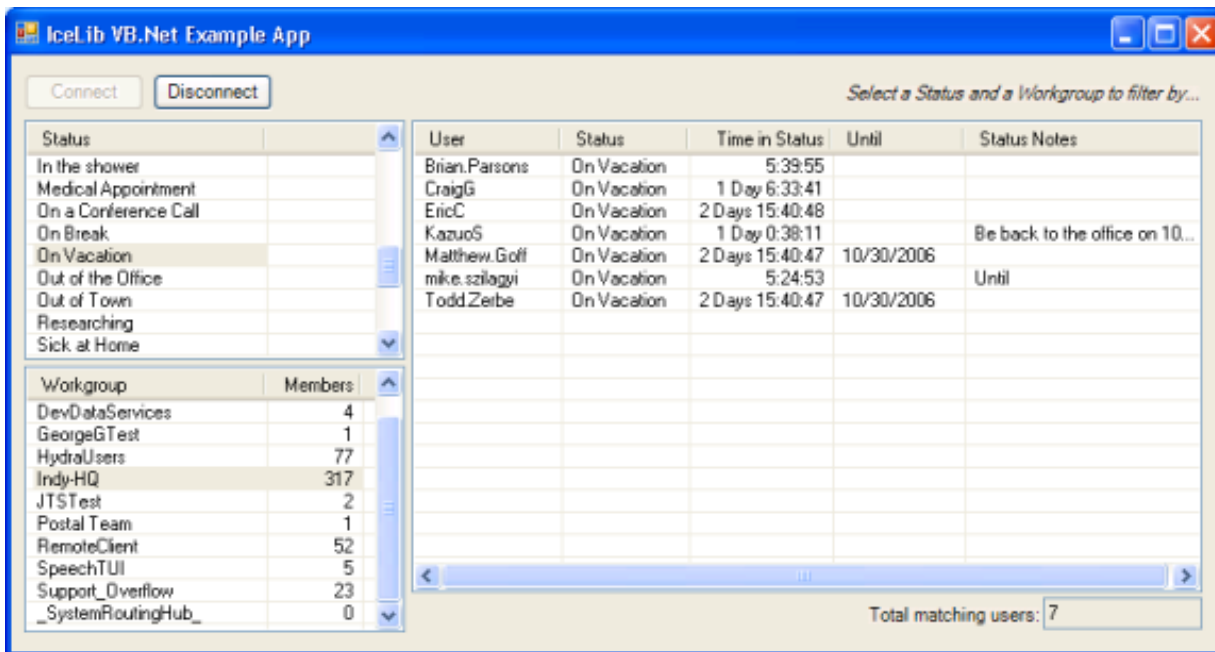
AspIceLibClientDemo ASP.NET Example

This example implements a web page to set status information and place a call. It demonstrates how to use the IceLib public API to do tasks that are commonly carried out by the various CIC Client applications. Click on **Create my Session** to start. If the link to create your session is unavailable then one currently exists.
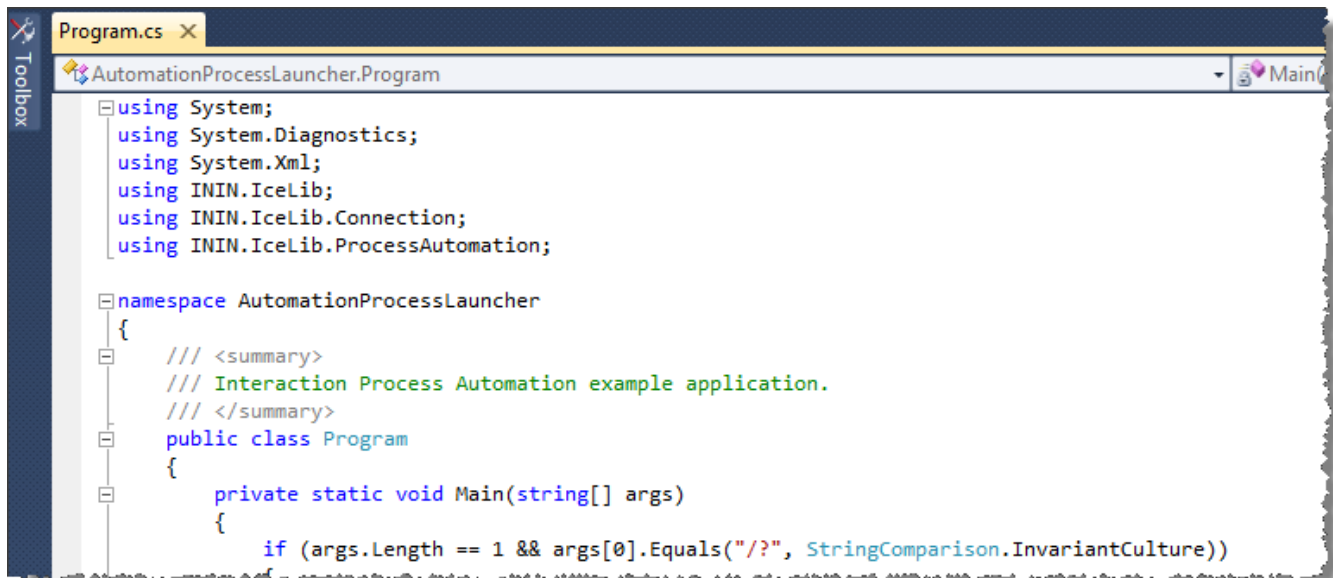

Configuration Cmdlet

This example application lets you use the IceLib SDK to configure a CIC server. It is a set of Windows PowerShell Cmdlets which are registered through the use of the SnapIn. The readme file gives full information about the cmdlets. You type commands on the command line in the Powershell window. In summary, cmdlets are:

- `New-Session:` Creates a session and connects it to the CIC server you specify.
- `Get-Session:` Retrieves and displays a list of active sessions.
- `Remove-Session:` Destroys the session(s) you specify.
- `Get-User:` Searches the CIC server for one or more users, then displays a list of results.
- `New-User:` Creates a new CIC server account with user information you specify.
- `Remove-User:` Permanently deletes a user account from the CIC server.
- `Set-Password:` Configures the password for a user or group of users.

The application also includes an example of how to format IceLib classes in Powershell and two simple example scripts: `ListAllUsers` and `BulkImport`.

AutomationProcessLauncher

C# source in the `ExampleApps\AutomationProcessLauncher` folder demonstrates the features of the ININ.IceLib.ProcessAutomation namespace. This is a simple console-based application that demonstrates launching an Interaction Process Automation process.

```csharp
using System;
using System.Diagnostics;
using System.Xml;
using ININ.IceLib;
using ININ.IceLib.Connection;
using ININ.IceLib.ProcessAutomation;

namespace AutomationProcessLauncher
{
    /// <summary>
    /// Interaction Process Automation example application.
    /// </summary>
    public class Program
    {
        private static void Main(string[] args)
        {
            if (args.Length == 1 && args[0].Equals("/?", StringComparison.InvariantCulture))
```

Change Log

The following table lists the changes to the *Introduction to IceLib Technical Reference* since its initial release.

Date	Changes
07-October-2011	Retitled document and updated with new information about the structure of the IceLib API help, as well as about each namespace and class.
28-June-2013	Updated copyright page and added 64-bit support.
10-July-2013	Added text to make it clear that 64-bit support is available in IC 4.0 Service Update 4 and later versions.
10-September-2013	Added new section on the namespace `Reporting.Interactions.Details`.
09-September-2014	Updated documentation to reflect changes required in the transition from version 4.0 SU# to CIC 2015 R1, such as updates to product version numbers, system requirements, installation procedures, references to Interactive Intelligence Product Information site URLs, and copyright and trademark information.
17-September-2015	- Updated documentation to reflect the addition of two CIC client applications, Interaction Desktop and Interaction Connect, and the removal of Interaction Client .NET Edition. - Updated document formatting.
25-April-2017	Updated documentation to reflect the removal of Interaction Client Web Edition and Interaction Client Mobile Web Edition.
07-September-2017	- Updated cover, copyright and trademark pages. - Applied Genesys terminology.
21-June-2019	Reorganized the content only, which included combining some topics and deleting others that just had an introductory sentence such as, "In this section...".