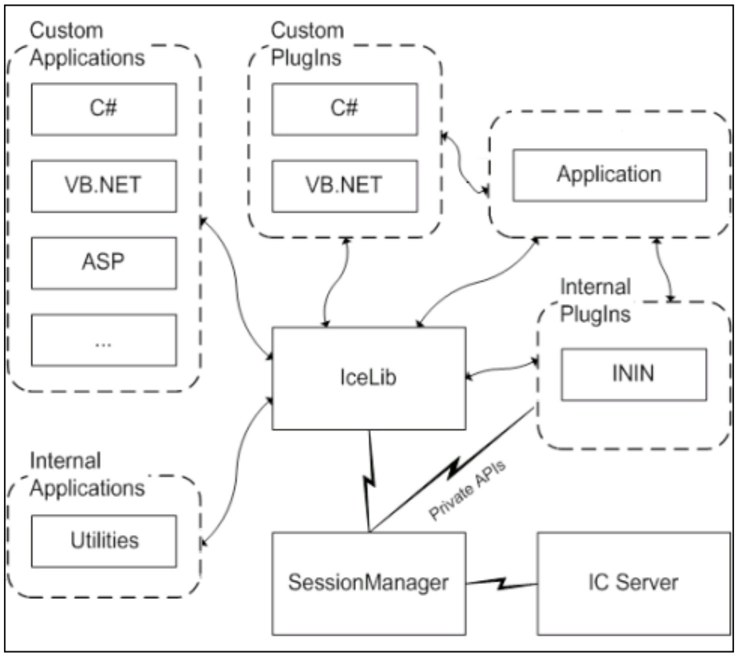
Introduction to the IceLib API

The *Interaction Center Extension Library* (IceLib for short) is a programming API that allows developers to create custom applications that leverage the Interaction Center 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 IC servers using the login options that are available in Interaction Client .Net Edition (e.g. user, password, station, remote number, remote station, persistent, audio enabled, etc.). The figure below illustrates the tight coupling between IceLib and the 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 IC Servers. Conversely, IceLib integrations for CIC 4.0 or later cannot be used with 3.0 IC Servers.

IceLib interfaces with *SessionManager,* the CIC subsystem that brokers connections between client applications and a given IC Server. Custom IceLib applications fully leverage SessionManager, just like internally developed CIC applications. For example, Interaction Client .Net Edition, Interaction Client Web Edition, Interaction Fax, and Interaction VoiceMail all 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 IC 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 Client .Net Edition 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.

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.

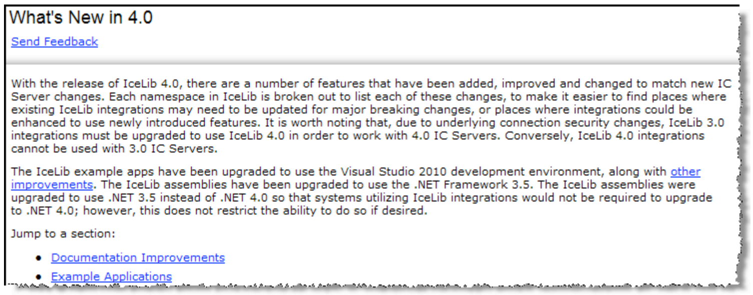
IceLib ships with full-featured sample applications in several languages (C#, VB.NET, ASP.NET). Each application is commented and designed to illustrate "best practices" of a real implementation. The examples cover key topic areas, such as:

- 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

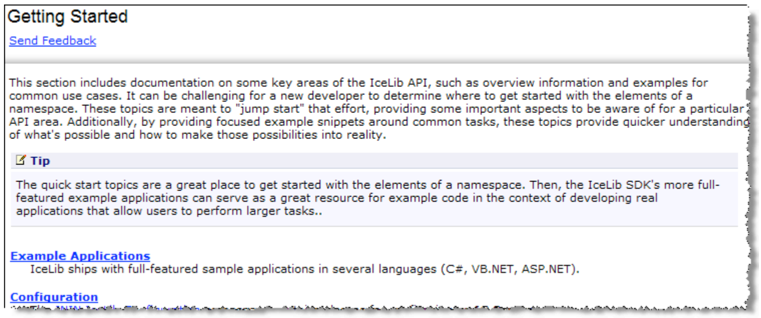
With the release of IceLib 4.0, there are a number of features that have been added, improved and changed to match new IC Server changes. Each namespace in IceLib is broken out to list each of these changes, to make it easier to find places where existing IceLib integrations may need to be updated for 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.

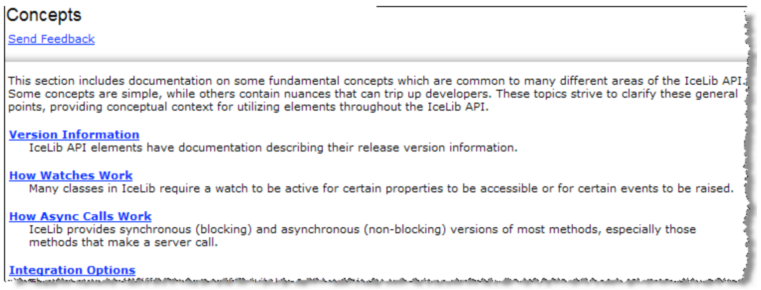
Tip

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
- ININ.IceLib.Configuration
- ININ.IceLib.Configuration.DataTypes
- ININ.IceLib.Configuration.Efaq
- ININ.IceLib.Configuration.Feedback
- ININ.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
- ININ.IceLib.Configuration.Validators
- ININ.IceLib.Connection
- ININ.IceLib.Connection.Extensions
- ININ.IceLib.Data.TransactionBuilder
- ININ.IceLib.Directories
- ININ.IceLib.EFaq
- ININ.IceLib.Interactions
- ININ.IceLib.People
- ININ.IceLib.People.ResponseManagement
- ININ.IceLib.ProcessAutomation
- ININ.IceLib.QualityManagement

- ININ.IceLib.Reporting
- ININ.IceLib.Reporting.Interactions
- ININ.IceLib.Reporting.Interactions.Filters
- ININ.IceLib.Statistics
- ININ.IceLib.Statistics.Alerts
- ININ.IceLib.Tracker
- 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 an IC 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 ...