INTERACTIVE INTELLIGENCE

Deliberately Innovative

Third-Party Integration

Technical Reference

Interactive Intelligence Customer Interaction Center® (CIC™)
Messaging Interaction Center™ (MIC™)

Version 4.0

Last Updated 2/21/2013

Abstract

This document is for developers and technical managers who want to understand how the Interaction Center can be extended programmatically to accommodate unique business needs.

Interactive Intelligence Inc.
7601 Interactive Way
Indianapolis, Indiana 46278
Telephone/Fax (317) 872-3000
www.ININ.com

Table of Contents

Third-Party Integration Technical Reference

Introduction

Virtually every office or call center has unique requirements that call for customized processing of transactions and customer interactions. The Interaction Center platform provides tools and application programming interfaces (APIs) that help customers create dynamic telephony applications.

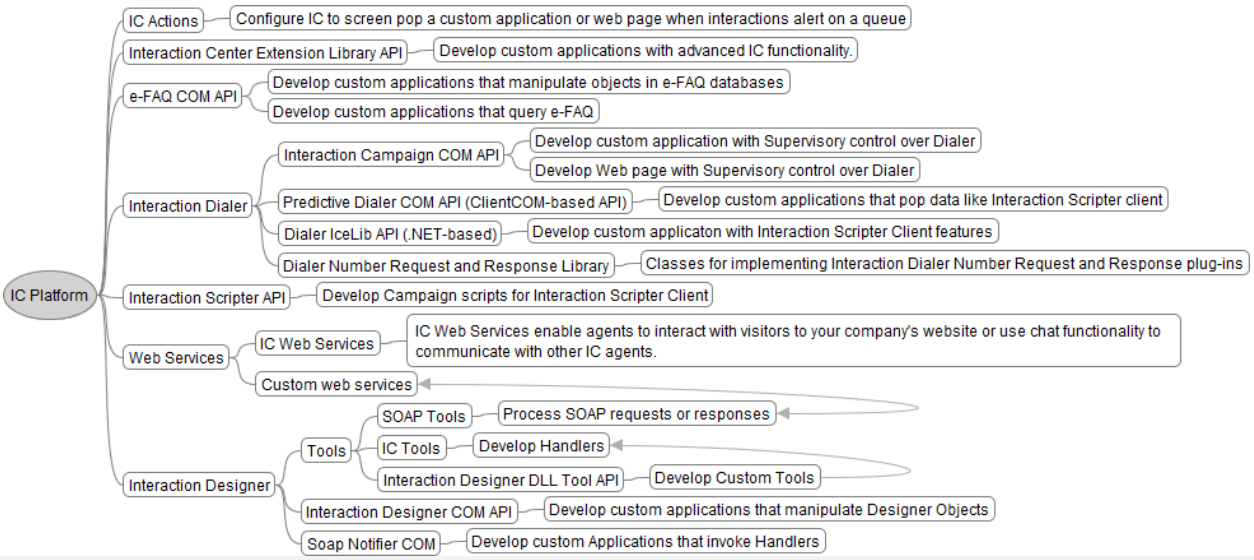
A growing number of APIs based on .NET or the Common Object Model (COM) deliver robust third party integration with IC, tightening integration of desktop applications with the IC server. The IC's object- and message-oriented design makes the system extensible at both the server and desktop levels.

Customers can develop applications that add or extend telephony functionality at the desktop, server, or web service level. This document describes development aids that make third-party integration possible. It is written for developers and technical managers who want to understand how the IC can be extended programmatically to accommodate custom business requirements. It is helpful to read this document before exploring developer-oriented publications that provide low-level programming details.

Since customers have many options at their disposal, this discussion is grouped into these major categories:

- IC Server Extensibility (p. 5)
- Desktop Application Extensibility (p. 10)
- Web Service Extensibility (p. 65)

IC Platform Extensibility Options



Integration Activities and Corresponding Software Support

Use links below to locate topics that most interest you.

Developer Requirement	Supporting Software
Create custom .NET application with robust IC functionality	Interaction Center Extension Library (p. 11) (IceLib) – the recommended API for custom IC integration.
Create handlers that provide precise control over interactions in IC	See Interaction Designer (p. 5) —the visual tool for writing telephony applications, without writing code.
Develop your own custom tools for developing handlers	See Interaction Designer DLL Tool API (p. 7)
Process SOAP requests and messages	See SOAP Tools in Interaction Designer (p. 67)
Write a program that manipulates handler objects	See Interaction Designer COM API (p. 7)
Write a program that can invoke handlers	See SOAP Notifier COM API (p. 70)
Write a progam that manipulates e-FAQ objects	See e-FAQ COM API (p. 26)
Pop an application or web page when interactions alert on a particular queue	By configuring actions in Interaction Administrator, you can easily screen pop an application on the agent's desktop, or open a web page when an interaction alerts on a specified workgroup or user queue. See DDE-based screen pop (p. 10).
Develop scripts that guide agents through stages of a campaign call	Interaction Scripter API (p. 64)
Develop custom applications or web pages that provide limited Supervisory control over Interaction Dialer	Interaction Campaign COM API (p. 57)
Develop custom applications that pop like Interaction Scripter client	Predictive Dialer COM API (p. 61)
Develop custom application with features similar to Interaction Scripter client	Dialer IceLib API (p. 64)

IC Server Extensibility

Telephony systems can be notoriously difficult to customize. "One size fits many" solutions limit your ability to customize interactions with customers. The high price of proprietary hardware, which is inevitably bundled with high-priced service contracts, makes it difficult to affordably differentiate your company's service offerings from your competitors.

Fortunately, the Interaction Center (IC) provides the integrated set of tools you need to customize its packaged modules. You can add new functionality to this already robust system, and you can integrate your own custom applications with IC.

Solutions based upon proprietary hardware and software often require specialists to manage the system. Even simple changes can be costly and cumbersome. The IC helps you avoid the hassles and expenses associated with integration of proprietary systems, by providing tools and APIs that seamlessly integrate new functionality into a system that was designed from the ground up to be extensible.

Interaction Designer—a visual tool for creating telephony applications

Interaction Designer is a visual programming environment for Interaction Center that makes it easy to create sophisticated server applications *without* writing code. Interaction Designer is powerful. IC's

default event-processing behaviors were created using Interaction Designer. Interaction Designer empowers any trained entry-level programmer to change the functionality of an IC system.

Electronic *interactions* between customers and businesses can take many forms. For example:

- Sending and receiving email
- Sending and receiving faxes
- Sending and receiving voice mail
- Simple and multi-party telephone calls
- Internet-based calls and chat sessions
- Remote access to voice mail, email, and faxes
- Intelligent call routing (ACD)

Each interaction is triggered by an *event* detected by IC, which, in turn uses an *initiator* to launch a *handler* to respond to the event and process it appropriately.

A *handler* is a logical flow of actions authored by an administrator or developer using Interaction Designer (ID). These actions are composed of small steps that do work of some sort. The author of a handler creates steps by dragging them off of a tool palette. Tools are template definitions of steps.

The IC includes predefined initiators to cover all of the most common events detected in business interactions. Interaction Designer, the premier development tool bundled with IC, makes it easy to create seamlessly integrated telephony applications.

Interaction Designer uses a *visual programming* metaphor. Traditional programming tools require developers to construct statements in a programming language. Interaction Designer permits developers to visually lay out logic needed to respond to a given event, by arranging and connecting graphical objects on the screen. Each object performs a specific task, such as sending a fax, e-mail, or routing a call.

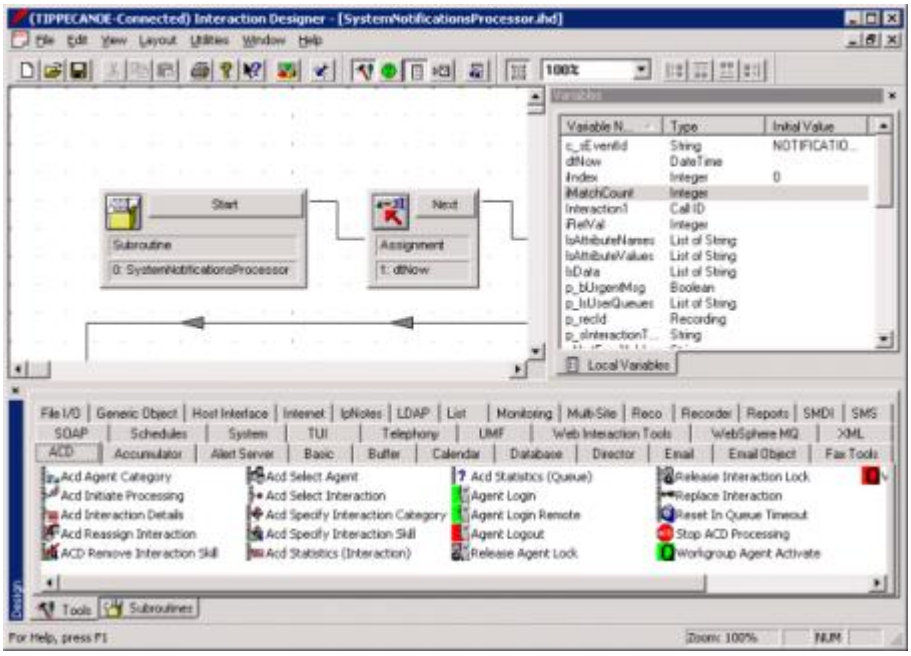


Figure 1. Telephony applications are created in Interaction Designer by arranging tools steps in a logical sequence.

Interaction Designer creates event-processing programs called *handlers*. Handlers encapsulate program logic within graphically connected nodes, called *steps*, for events processed by the IC server. Each step is created by dragging a *tool* (a template definition of each step) from a palette, pasting it in the workspace, and editing the tool's properties. Interaction Designer provides an extensible palette of tools that serve as building blocks for building handlers.

The first step in each handler is an *initiator*. An Initiator identifies the event that starts a handler. An example of an event might be 'Incoming Call'. When the Interaction Processor receives a notification that an 'Incoming Call' event occurred, it starts the handler whose initiator is configured for 'Incoming Call'.

When you finish building a handler, you can elect to have Interaction Designer *publish* it. Publishing a handler converts your visual design to Java source code. (Absolutely no knowledge of Java is required. Visually designed logic is automatically converted to Java program code that is compiled to create a highly efficient server application.)

The general process of building handlers can be broken down into these six steps.

1. **Research**. Start by becoming familiar with the handlers and subroutines currently running on your IC system.
2. **Design**. Map out the functionality of the handler you plan to build, and figure out how it might interact with the other handlers and subroutines.
3. **Build**. Create and link steps to define new functionality, using Interaction Designer.
4. **Publish**. The automated Publish process compiles your handler and moves it to the server.
5. **Test**. Test the handler on your live system or on a second system you've set up for testing purposes. We highly recommend the latter.
6. **Activate.** If you are not already using the handler on your live system, activate the handler.

When you publish a handler, events configured for its initiator are registered with the Interaction Processor. Interaction Processor then instructs the Notifier that it wants to be notified any time that event occurs. Whenever the event occurs, Notifier tells the Interaction Processor, and Interaction Processor starts a new instance of the handler. The active lifetime of a handler is very short, since handlers terminate after processing an event.

Sometimes, modifying an existing handler is the best way to implement new functionality. For example, you might want to add a new key-press option for callers using your IVR (Interactive Voice Response) system. In this case, adding new functionality involves recording a new prompt, adding a condition to a Selection step, and adding additional steps to your existing IVR handler. You don't need to write a new subroutine or handler to add simple functionality. To make minor changes or additions to the IC, it is often best to modify existing handlers or subroutines. For more information about Information Designer, refer to *IC Interaction Designer Help*.

Interaction Designer DLL Tool API

You may be wondering, "What if I need functionality that Interaction Designer doesn't include? Can I create my own tools, and use them with Interaction Designer?"

The answer is yes. Interaction Designer DLL Tool API provides the means to create tools for any type of situation. Your new tools can be added to Interaction Designer's standard tool palette.

Interaction Designer DLL Tool API allows a DLL to be registered as a new Interaction Designer tool. Tools are used to create handlers for the IC system. Each tool corresponds to a function in a DLL. Tools are template definitions for steps in a handler. Your tool DLL may contain business logic that is unique to your organization, or it may perform operations on data collected by agents in a call center. Your custom DLL might process a credit card transaction, for example.

To use this API, you must be familiar with C programming and with the basic concepts of Interaction Designer, handlers, and the handler development process. Handler developers can then use your custom tool to build handlers with virtually unlimited functionality.

The Interaction Designer COM API

The *Interaction Designer COM API* gives developers programmatic control over the familiar visual objects exposed by Interaction Designer. This API provides object-oriented access Designer "objects", such as handlers, tools, subroutines exit paths, and aspects of Designer itself.

Developers who are familiar with handler development and with a programming language that supports the component object model can use this API to create applications that integrate with Interaction Designer.

As mentioned earlier, Interaction Designer is the visual programming environment for the IC platform that creates sophisticated server applications. Designer is powerful—all of the Interaction Center's default event-processing behaviors were created using Interaction Designer, *without* writing code.

The *Interaction Designer COM API* makes it possible to manipulate objects in Interaction Designer by writing procedural code, using traditional programming languages such as C++ or Visual Basic 6.

New objects, such as handlers, can be created on the fly. The API is a gateway into Interaction Designer that provides access to tools, handlers, steps, messages, subroutines, initiators, exit paths, and to properties of Designer itself. This API is compatible with Windows programming languages that support COM.

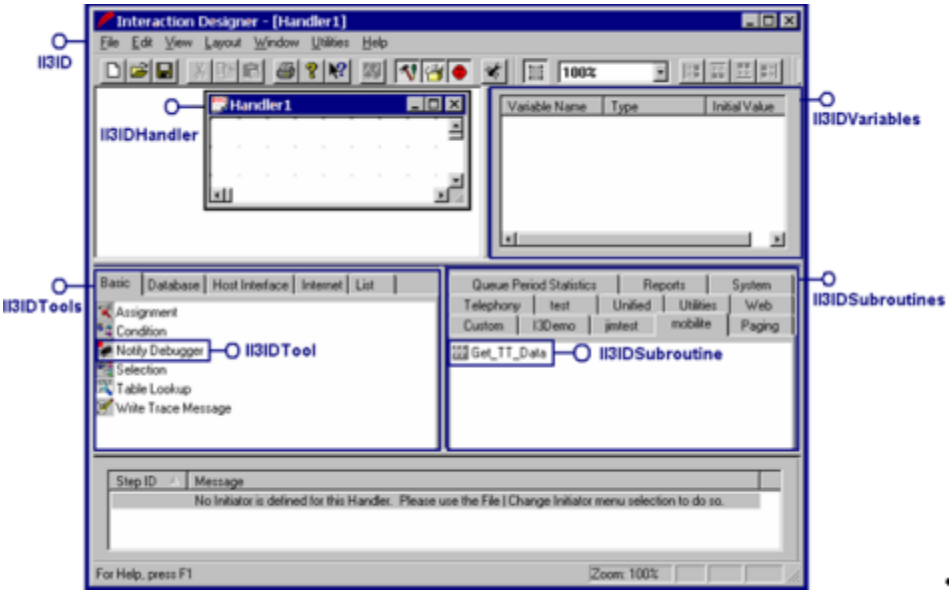
Designer COM Objects

In Interaction Designer, handler developers lay out logic that responds to a given event by arranging and connecting on-screen graphical objects. Each object performs a specific task, such as sending a fax, e-mail, or routing a call. These event-processing programs are called *handlers*.

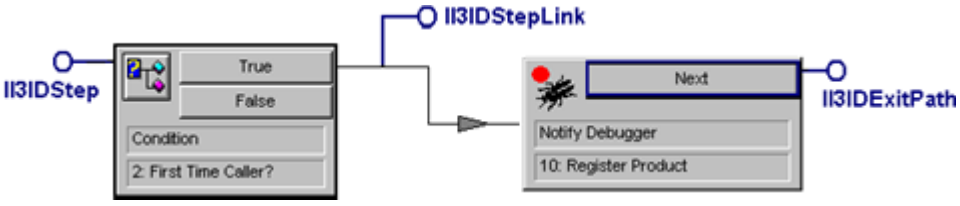
Handlers encapsulate program logic within graphically connected nodes, called *steps*, for events processed by the IC server. Each step is created by dragging a *tool* (a template definition of each step) from a palette, pasting it in the workspace, and editing the tool's properties. Interaction Designer provides an extensible palette of tools that serve as building blocks for building handlers.

The first step in each handler is an *initiator*—it identifies the event that starts the handler. An example of an event might be 'Incoming Call'. When Interaction Processor receives a notification that an 'Incoming Call' event has occurred, it starts the handler whose initiator is configured for 'Incoming Call'.

Interaction Designer *publishes* handler logic. This process converts visually designed logic to Java program code that is compiled to create a highly efficient server application. Most non-visual objects (supported by the API) are analogs of familiar visual objects in Interaction Designer. For example, a handler object in the API provides access to a collection of step objects. The image below shows the correlation between visual elements in Interaction Designer and objects exposed by the API:



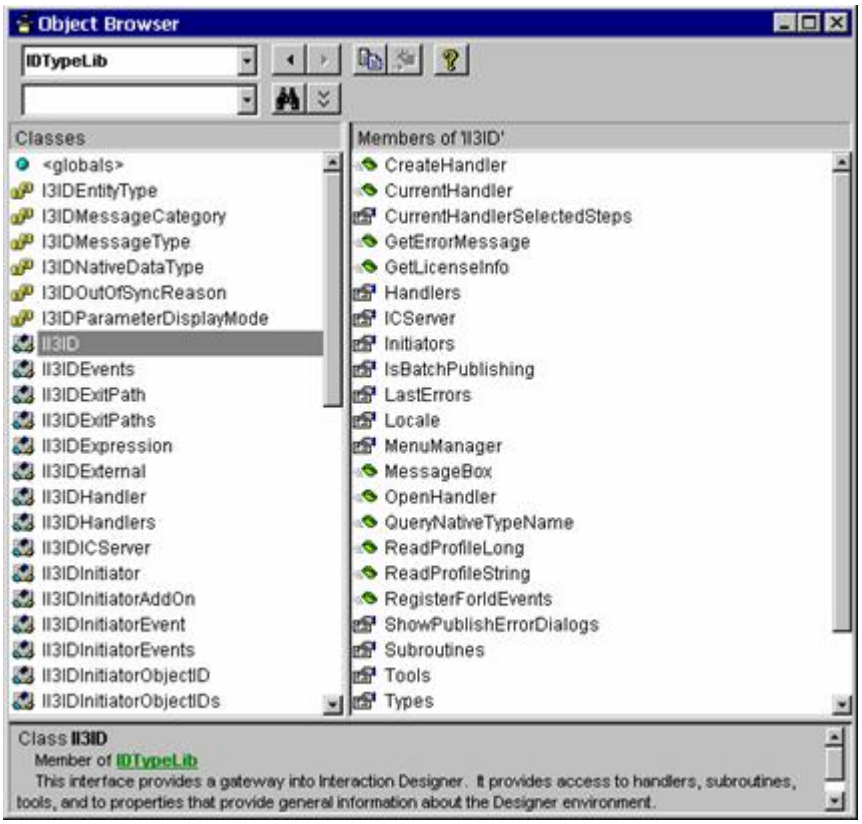
Interfaces in the Designer COM API provide access to familiar objects such as handlers, tools, subroutines, or variables, and even to some aspects of Designer itself. Each step has properties such as the step name, position, and exit paths that can be managed programmatically. Likewise, you can call methods that perform actions on Designer objects. For example, you can publish a handler by calling the Publish method of a handler object, or open a handler by calling the OpenHandler method of the Designer object.



The II3IDStep, II3IDStepLink and II3IDExitPath interfaces wrapper tool steps, exit paths, and linkages between steps.

Interaction Designer must be running when Designer COM applications are executed.

This API provides a comprehensive set of classes that wrapper tools, handlers, steps, messages, subroutines, initiators, exit paths, and properties of Designer itself. New objects (handlers, steps, etc.) can be created and published under program control, and existing objects can be enumerated and modified on the fly.



Exploring Designer COM API classes using the Visual Basic 6 Object Browser

For more information about Interaction Designer COM API

Refer to the *Interaction Designer COM API Developer's Guide*. You'll find this publication in the *System APIs* section of the *IC Documentation Library* on your IC Server. It provides a complete reference to objects, methods, and properties.

Desktop Application Extensibility

In this section, we discuss techniques that customers can use to extend the functionality of desktop applications, ranging from simple *screen pop* to complex APIs that developers use to create desktop applications. A common need is to launch an application in response to a communications event, such as a telephone call. The telephony industry term for this is "screen pop".

DDE-based screen pop

The simplest way to pop an application is to use *Dynamic Data Exchange* (DDE), an inter-process communication method that allows applications to exchange data and commands. DDE actions can be configured in the Interaction Center to launch an application on an agent's desktop when a call (or other communications event) alerts for that agent in *Interaction Client .Net Edition*.

Interaction Client .Net Edition is the "software phone" application that adds the sophistication of executive telephones to any telephone handset. Interaction Client makes it easy to answer, record, conference, place, and transfer calls. Businesses and call centers use Interaction Client to manage customer interactions at the desktop.

Most Windows business applications provide a DDE interface. For example, Interaction Client can launch Excel, Word or any DDE-compliant custom application on an agent's desktop when a call alerts for that agent.

DDE is based on the concept of *client* and *server* applications. The third-party application that sends data is known as the DDE *server*. The application that receives and manipulates data (or commands) from the server is referred to as the DDE *client*.

- Interaction Client .Net Edition functions as a DDE client only. It cannot be used as a DDE server. Interaction Client .Net Edition can pass messages to invoke other DDE-enabled applications when a call alerts, disconnects, or is transferred from an agent's station, but it cannot accept commands from applications.
- The data passed from IC to the DDE server (third party application) can include the caller's ID and any other attribute information associated with that interaction.

No programming is required to take advantage of Interaction Client's built-in DDE support. To pop applications when calls arrive on a workgroup or user queue, the requirements are:

1. *Interaction Client .Net Editio*n must be running on the agent's system. It can be minimized.
2. The application to "pop" must conform to Microsoft DDEML specifications. In other words, it must support DDE.
3. The application to pop must be installed on the agent's workstation in the *System Path*.
4. The application to pop must be defined as an action in Interaction Administrator.
5. The DDE action must be registered with a specific user or workgroup queue. That tells IC to notify Interaction Client when it should pop the application in response to a communications event, such as a call alerting on the queue.

The ability to pop familiar application software saves time, money, and internal training costs. Generally speaking, DDE integration with Interaction Client conserves resources, especially for agents who are familiar with a particular customer support application.

Should I use IceLib instead of DDE?

Applications need not use DDE to pop an application in response to an incoming call. Similar functionality can be provided by IceLib (p. 11). The choice is up to you.

One difference between DDE and the other APIs is that DDE applications use Interaction Client to start another application and then pass call attribute information into it. IceLib does not use or require Interaction Client. It is worth noting that IceLib requires a license purchase. DDE is free.

Significant programming expertise is required to develop IceLib applications. If your organization is supported by a staff of experienced programmers, then IceLib may suit your needs. However, if simple screen pop is all you need, the DDE approach provides a simple and reliable mechanism that is easily implemented. IceLib is discussed later in this document.

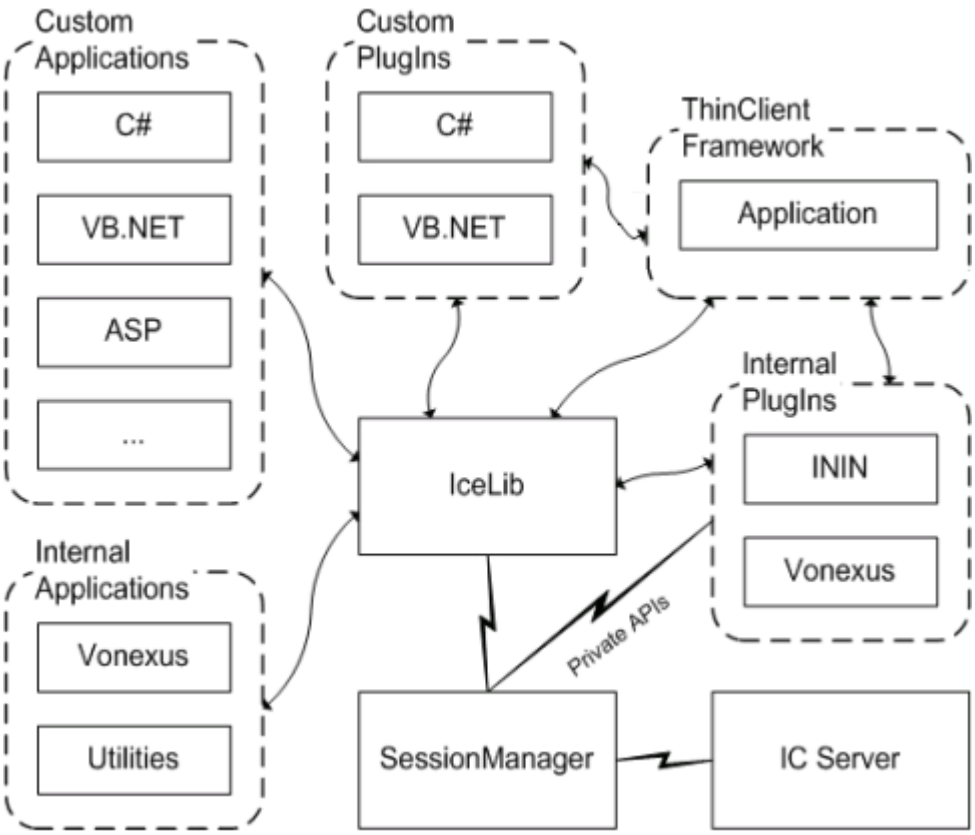
For more information about DDE:

See *Interaction Client .NET Edition DDE API Developer's Guide*. You'll find this publication in the *System APIs* section of the *IC Documentation Library* on your IC Server. It explains how to define and register actions, and provides other details about DDE.

Interaction Center Extension Library (IceLib)

Interaction Center Extension Library (IceLib for short) is the premier development tool for desktop application development. This uniquely modern API is for developers who use modern .Net languages, such as C# or VB.Net.

IceLib provides a clean architecture that applications and ThinClient plug-ins can use to manage sessions with IC. 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 to ThinClient applications (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.



IceLib interfaces with *SessionManager,* the IC subsystem that brokers connections between client applications and a given IC Server. Custom IceLib applications fully leverage SessionManager, just like internally developed IC applications. For example, Interaction Client .Net Edition, Interaction Client Outlook 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 to 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 ThinClient feature sets to third-party applications. It provides a stable foundation for ThinClient plug-ins and provides support for stand-alone applications. 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#)

IceLib Namespaces

This API is encapsulated in a single root namespace (IceLib), located directly off of the ININ namespace. IceLib's 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 (p. 13)
- ININ.Icelib.Connection (p. 13)
- ININ.Icelib.Data.TransactionBuilder (p. 22)
- ININ.Icelib.Directories (p. 14)
- ININ.Icelib.Interactions (p. 16)
- ININ.Icelib.People (p. 17)
- ININ.Icelib.People.ResponseManagement (p. 18)
- ININ.Icelib.Tracker (p. 19)
- ININ.Icelib.UnifiedMessaging (p. 21)

ININ.Icelib Namespace

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.

ININ.Icelib.Connection Namespace

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 (IC 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).

ININ.Icelib.Directories Namespace

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 IC 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). The main elements are listed below.

Directories Manager

The Directories Manager is the key object to access all APIs for directory support. It's 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 IC 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 directory 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 IC this is mostly supported in the speed dial and general contact directories.

Watched View Support

The watched view provides a way to performing '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.

ININ.Icelib.Interactions Namespace

This namespace represents the functionality of Interactions, Queues, and Conferences. It provides classes that manipulate Interaction Center interaction queues. For example, you can make a call, create a "Sidebar Chat" (i.e. a conference chat), or initiate an Agent Help Request.

1. **Interactions** support *attributes*, *watches*, and *actions*. The Interaction types are Calls, Chats, Emails, Callbacks, Recorder, and Monitor objects.

   The **attributes** of an Interaction are queried by examining strongly-typed public properties. For example, to see if a call is muted, you can check its IsMuted property. User-defined attributes can be queried by name.

   Watches generate an event notification when an attribute changes in value.

   Actions do something to an Interaction. If its type allows, an Interaction can be:

Transferred	Held
Muted	Picked up
Joined	Sent to voicemail
Listened	Coached
Recorded (and have recording paused)	Marked private
Have help requested	Set an account code Id
Set a wrap up code Id	Set a remote name
Set raw string by name (if allowed)	Receive events when attributes change value

   As mentioned earlier, the operations that can be performed on an Interaction vary by interaction type. The table below illustrates these differences:

Interaction Type	Supported Operations
Queues	Watch for attribute changes for Interactions on a Queue. Receive events when Interactions are added/removed from a Queue. Receive add/change/remove events for Conference interaction changes.
Call	Perform standard Interaction operations. Make a call Receive events when calls are made Play digits
Chat	• Perform standard Interaction operations. • Create chats • Get chat members. Chat members have: type (internal/external, display name, associated interaction id, "typing indicator") . There is an explicit "System Member" that represents the IC system chat text. • Add chat members • Receive chat membership events • Send: Text, Url or File • Receive Text or Url added events • Set a "typing indicator" • Receive "typing indicator" events for chat members
Email	• Perform standard Interaction operations. • Get the response • Set the response

- Sender
- Subject
- Body
- To, cc, and bcc recipients
- Original and new attachments

- Download attachments (original and new)
- Upload attachments
- Remove (new) attachments
- Receive events when the response is updated
- Send the email
- Resolve email address names
- Mark the email as "spam" (if an Account Code has been assigned for spam)

Conference
- Create conferences
- Get conference parties
- Add conference parties

Agent Help Request
- Receive events for new help requests
- Receive events for new help responses
- Send a help response

2. **Queues** support *watches* and provide access to *contents*. The contents of a queue are the Interactions that it contains. Your application can set up watches to receive notification events when queue *membership* changes (interactions added/removed), and when queue *attributes* change.
3. **Conferences** provide access to contents, operations, and watches. A custom application can be notified when queue *conference membership* changes (conference interactions added/removed), and when the *attributes* of a queue conference interaction change.

ININ.Icelib.People Namespace

The ININ.IceLib.People namespace contains classes for accessing 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 to 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 an IC user.
- UserAccessListsSettings—the access control lists settings for an IC user.
- UserDataSettings—miscellaneous data settings for an IC user, such as the user's workgroup membership, supervisory rights, and greeting preference.
- UserSettings—basic settings for an IC user and access to all watched attribute and object classes in the People namespace.
- WorkgroupDetails—the common details for an IC 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 an IC user.
- FilteredStatusMessageList—list of status messages available for an IC user.
- StatusMessageList—list of status messages defined in the IC system.
- UserWorkgroupActivationList—list of workgroup activations for an IC user.
- UserStatusList—the current status for a list of IC users.

ININ.Icelib.People.ResponseManagement Namespace

Response Management is the general term for features that allow a user to send pre-defined 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 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 the Interaction Client (or a custom IceLib application). The agent "picks up" the interaction request. Interaction Client 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 Client 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. For example, an Interaction Url titled 'Support web site' could be defined as 'http://www.inin.com/support'.

- An Interaction File points to a file path. For example, an Interaction File titled 'Icelib Documentation' could point to 'C:\Program Files\Interactive Intelligence\Icelib\Documentation\Icelib.chm'.

ResponseItemType indicates if a ResponseItem is a Message or a Url or a File or a 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.Tracker Namespace

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.

Any application that utilizes 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 2.0 *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 Namespace

This namespace manages *Faxes* and *Voicemails*. Unified Messaging classes allow access and manipulation of Fax and voicemail files generated by an IC server. Typically these files are delivered to a mailbox through the corporate Microsoft Exchange or Lotus Notes server, although Interactive Intelligence 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 IC 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 an IC 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 IC 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 being submitted to the IC 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 IC server.

Voicemail Capabilities

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

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.

ININ.Icelib.Data.TransactionBuilder Namespace

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.

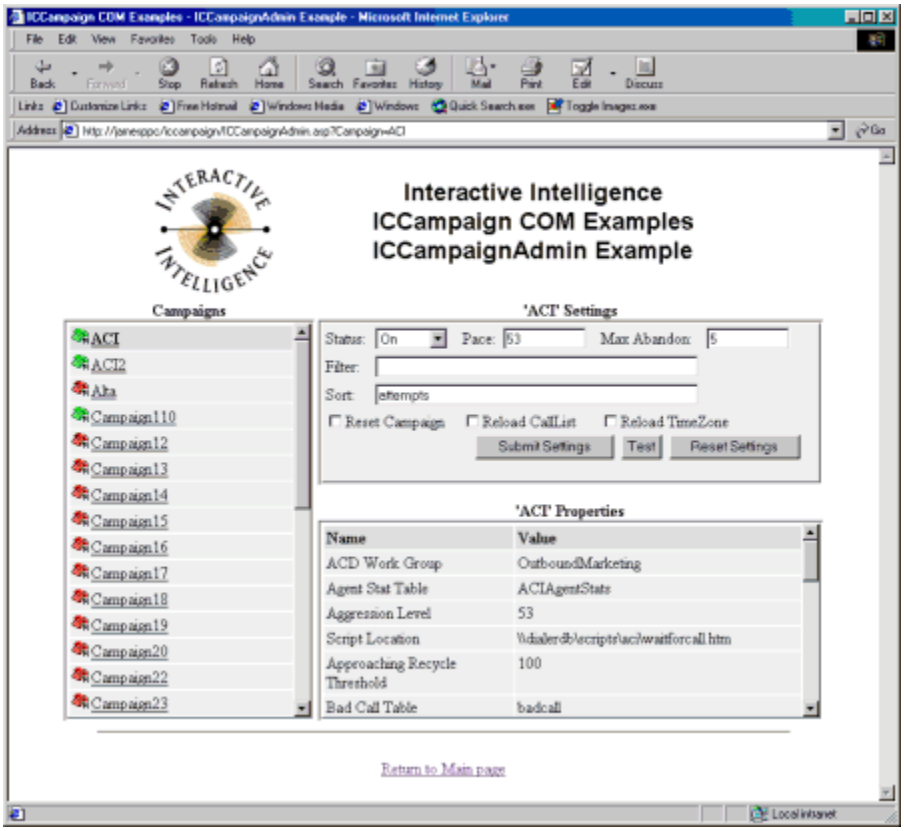
Other Interface-Based APIs

Without question, component software makes it easier to develop computer programs. The Interaction Center supports sophisticated software components that empower programmers to construct robust telephony applications. These components are compatible with any Windows programming language that supports Microsoft's Component Object Model.

1. Interactive Intelligence also develops COM components for its non-IC products, such as e-FAQ. The e-FAQ COM API (p. 26) wrappers objects in a "Frequently Asked Questions" database, called an FAQ. The components of this API provide programmatic control over addresses, keywords, aliases, entries, killwords, and other constituent parts of a frequently asked question. It also permits developers to manipulate queries, entries, and query results.

2. The Interaction Campaign COM API (p. 57) empowers developers to create custom applications or web pages that provide limited *campaign administration* functionality in an Interaction Dialer environment. Interaction Dialer is a predictive dialing add-on for IC that enhances the efficiency of outbound call centers. Interaction Dialer automates many manual tasks that agents perform. For example, it retrieves a telephone number from a call list file, appropriates an outbound line, dials the number, and waits for the call to be answered before routing the connected party to an agent. Dialer detects ring/no-answer conditions, busy signals, FAX tones and answering machines. Agents receive only calls that have reached a targeted party.

   Dialer includes Interaction Campaign Administrator, a robust utility that provides call center administrators with the complete functionality needed to define and administer campaigns. In some circumstances it is desirable to limit the level of administrative control that personnel can exert over Interaction Dialer. For example, a night supervisor who needs to start or stop campaigns does not necessarily need the ability to define campaigns, or modify schedule settings. The Interaction Campaign COM API makes it possible to develop custom applications or web pages that provide appropriate supervisory control over the Dialer.



   Interaction Campaign COM ships with several sample applications. This web page example provides the control needed to administer a campaign without exposing the advanced functionality of Interaction Campaign Administrator.

3. The Predictive Dialer COM API (p. 61) enables programmers to develop applications that are functionally similar to the Interaction Scripter client. It provides a *callback object* that notifies an application when predictive events occur, and a *server object* whose methods manipulate and disposition data pops. The server object logs in the agent, transitions to the next stage, and updates contact list data. It provodes overall control over interactions between an agent and the Outbound Dialer server.
4. Other COM components manage interactions between the IC and external servers. For example, SOAP Notifier COM Components (p. 70) allow the IC integrate with external network devices in an IP environment. SOAP is a wire protocol that allows dissimilar servers to exchange information and to execute commands.

About the Component Object Model

Component Object Model (COM) is the latest in a series of software standards developed by Microsoft. The COM specification explicitly defines how binary software components may interact with one another. COM also defines the format of compiled machine code, irrespective of the programming language used to create a COM component.

Since COM is a binary standard, it removes compatibility barriers normally associated with the development of reusable software components—at least for Windows users. For example, a Delphi programmer can use COM components that were developed by Java, C++, or VB programmers. Any of these languages can create COM components that are compatible with the others.

Component software has traditionally been targeted at specific programming languages. C++ programmers created VBX controls for Visual Basic programmers; Delphi programmers bought VCLs. However, the rationale behind component software has always been the same: to extend the functionality of software, without writing a lot of code, by "plugging-in" small elements of software.

In fact, component software has become mainstream, while *object-oriented* programming has not. COM provides the best features of both. COM implements object-oriented principles by encapsulating object-oriented, reusable components as binary entities, rather than as source code stored in language-specific class libraries. COM does this in a way that permits developers to update COM components, without breaking the applications that use them.

What is a COM Object?

In the world of COM, an *object* is an instance of software that contains the functions (called methods) that represent what the object can do (its intelligence) and associated state information (called properties) for those functions. An object is a data structure and functions that manipulate the structure.

COM Interfaces

In COM, the only way to manipulate data associated with an object is through an *interface* to *methods* in the object. Since developers can add new functionality without breaking legacy applications, COM overcomes compatibility issues traditionally associated with DLLs.

When a COM component is updated to add additional or improved functionality, a new interface is defined that provides access to new methods (functions) in the object. Programs written for the original interface continue to work, since the updated COM component still supports the original, unchanged set of interfaces and methods as it did before, in addition to its new interfaces and methods.

Once the public interface is defined, it can't be changed later. Instead, COM component developers define a new interface that provides access to methods in the component. Since each set of methods belong to one interface, dramatic changes can be made between releases of COM components without incurring compatibility problems.

COM objects have Methods and Properties

When a COM component is loaded into memory, its functions are mapped to internal memory addresses. A COM *interface* is simply a pointer to a location in memory where a set of functions is stored. Functions are called *methods* by COM programmers. Methods perform some sort of action on an object. For example, the *publish* method of a handler object publishes a handler

under control of the Designer COM API. Many COM object support *properties*. A property is an item of data associated with an object. For example, a handler object has a Name property.

By convention, COM interface names begin with the letter "I", which stands for "interface". All COM objects implement an interface named *IUnknown*, which exposes three functions:

- **AddRef**. The AddRef function increments an internal counter each time the object is referenced.
- **Release**. The Release function decrements an internal counter each time that an object is released. When the internal reference counter becomes zero, Windows deletes the object.
- **QueryInterface**. The QueryInterface function queries a COM object to find out if it supports a particular interface. If it does, QueryInterface provides a pointer to the set of functions exposed by the interface.

It is unlikely that you will ever need to use IUnknown to query an IC-related COM component, since our interfaces, methods, and properties are fully documented. However, each IC COM component supports the standard IUnknown interface and its methods. The IUnknown interface is not described in any API documentation, since IUnknown is defined by the COM specification and is always available.

COM Clients and Servers

As with DDE, there are COM *clients* and COM *servers*. However, in the world of COM, these terms have new meaning.

A COM *server* interacts with clients by implementing interfaces that direct the client to methods supported by the interface. *In-process* COM servers are DLLs that run on the local machine. *Out-of-process* COM servers are .EXE files that can reside on the local machine, or on a networked computer. The COM specification has been extended to provide support for *distributed* processes (DCOM) running on remote computers. The term COM is now synonymous with both COM and DCOM.

A COM *client* is a program or object that calls methods in one or more interfaces provided by a COM server.

COM is a refinement of the thinking behind OLE, DDE, and ActiveX. COM makes it possible to create truly reusable software components that are compatible across programming languages. COM provides some relief from version conflicts associated with software updates, and creates a platform for truly distributed network computing.

The e-FAQ COM API

e-FAQ is a product sold separately from the IC that instantly answers Frequently Asked Questions (FAQs) via e-mail and the Web. E-FAQ is ideal for companies that use CIC or MIC. e-FAQ employs advanced linguistic analysis and artificial intelligence techniques to examine inquiries, search for matches, and automatically respond when an appropriate match is found. e-FAQ also simplifies the authoring process with its Knowledge Manager interface, and provides Web site visitors an efficient "self-service" environment with escalation to live help.

A sample web page from which a query can be submitted to e-FAQ.

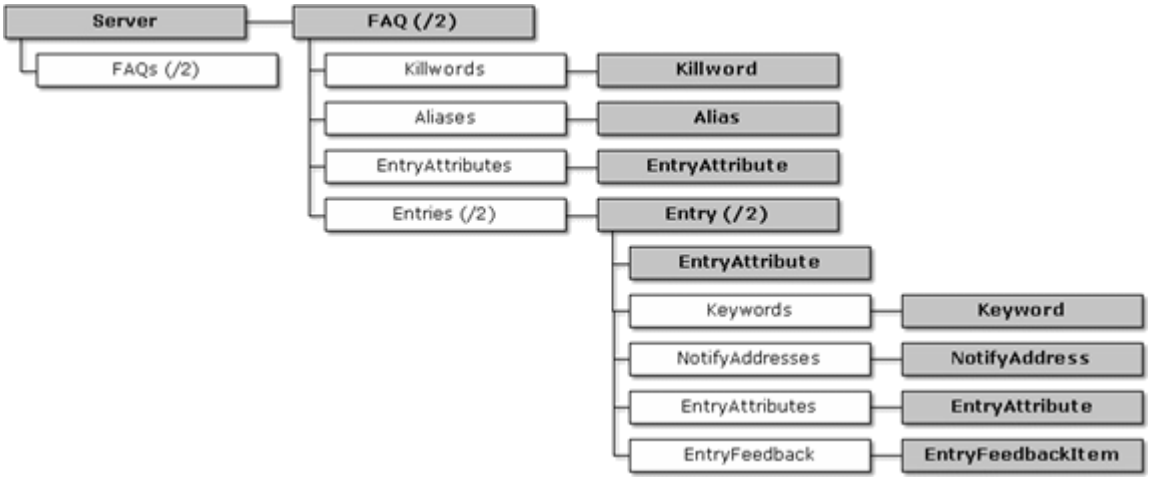
The e-FAQ COM API (p. 26) wrappers objects in FAQ databases. The components of this API provide programmatic control over addresses, keywords, aliases, entries, killwords, and other constituent parts of a frequently asked question. It also permits developers to manipulate queries, entries, and query results.

e-FAQ Objects

The diagrams below visually describe objects and collections in the e-FAQ COM API. Interfaces that manipulate objects are represented in the diagrams using a shaded background. Interfaces that manipulate collections of objects have a white background.

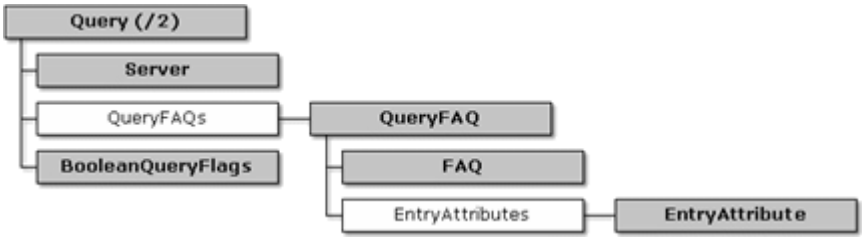
Server Interface Objects

Interfaces in the Server category manage addresses, keywords, aliases, entries, killwords, and other constituent parts of a frequently asked question.



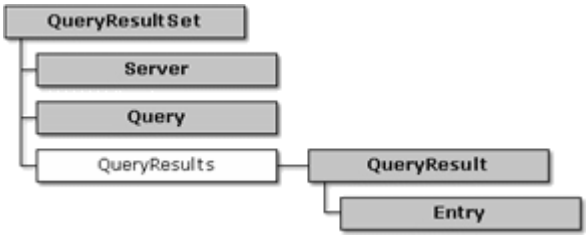
Query Interface Objects

Query interfaces define and execute a FAQ query.



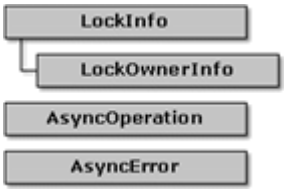
Query Result Set Objects

Query Result Set interfaces provide access to query result set objects. Each query result object represents one row in the result set. You can obtain statistics about an Entry selected as the result of a query, or enumerate a collection of query result objects.



FAQ / Entry Locking Objects

The objects you create using Locking interfaces provide information about the owner of a FAQ or Entry lock.



e-FAQ Interfaces

Interfaces in the e-FAQ COM API are listed below.

IAlias Interface

The objects you create using this interface denote an alias. An alias represents a word and its expansion. For example, you can add an alias that expands 'IC' to 'Interaction Center'.

Methods

ApplyChanges

This method sends the changes you made locally to an Alias to e-FAQ Server. This method fails with an error if the Alias is not dirty. See also: IAlias::IsDirty property.

Clone

This method allows you create a new alias by cloning (copying) the contents of an existing alias. The new alias is not automatically added to a FAQ. Use FAQ.Aliases.Add to add the new alias to a FAQ.

Revert

This method discards local changes made to an existing alias, and refreshes the alias object by reading the current state from e-FAQ server.

Properties

AliasId (Get)

The AliasID property returns the identifier of an alias. The AliasId is used internally to identify this alias. More precisely, it is the primary key in the I3EFAQ_F_Aliases table.

Applicability (Get)

This property indicates when an alias is to be expanded. The return value is an AliasApplicability constant that indicates that the Alias will be expanded during indexing, during querying, or at both times.

Applicability (Put)

This property determines when an alias will be expanded. You must specify one of the abovementioned AliasApplicability constants as an input parameter.

Expansion (Get)

This property returns a string containing the word or words that an alias will expand to. For example, if the alias is 'IC', the expansion property might return 'Interaction Center'.

Expansion (Put)

This property assigns the word or words that an alias expands to. For example, if the alias is 'MIC', the expansion property might return 'Messaging Interaction Center'.

FAQ (Get)

This property returns the FAQ object that owns the alias.

IsDirty (Get)

This property is True if the alias has been modified locally, meaning that changes have not been sent to the server yet.

IsNewAlias (Get)

This property is True if the alias is a new alias object, that has not been added to the Aliases collection of a FAQ.

Type (Get)

This property returns the type of alias, depending upon whether the alias is an abbreviation, acronym, or something else.

Type (Put)

This property sets the type of alias, depending upon whether the alias is an abbreviation, acronym, or something else.

Word (Get)

This property returns a word that can be expanded. For example, 'IC' can be expanded to 'Interaction Center'.

Word (Put)

This property assigns a word that can be expanded. Note: The word of an Alias cannot be changed once the Alias has been added to a FAQ.

IAliases Interface

This interface represents a collection of alias objects. You can return the count of items in the collection, retrieve an alias by specifying its index, add items, remove items, and enumerate the collection.

Methods

Add

This method adds an alias object to the collection.

Item

This method returns the Alias object specified by its index within the collection.

Remove

This method removes an alias from a collection of alias objects. To specify which item is removed, the function is passed an index number (1-based), or the string value of the alias itself.

Properties

Count (Get)

This read-only property returns the number of elements in a collection of alias objects.

_NewEnum (Get)

This property returns an IEnumVARIANT instance.

IAliases2 Interface

This interface extends the IAliases interface to add methods for searching and refreshing the list of aliases from the server.

Methods

Find

The Find method searches the Alias object given its AliasId. If a match is found, an IAlias object is returned. NULL is returned if no match is found.

Refresh

The Refresh method refreshes the list of Aliases from the server.

IAsyncError Interface

This interface supports an asynchronous operation error object that contains error information about a failed asynchronous operation.

Properties

Code (Get)

This read-only property returns the error code number.

Description (Get)

The Description property returns the descriptive text of the error that occurred.

NativeObject (Get)

This method returns a potential error object, or NULL if none is available.

IAsyncOperation Interface

This Interface supports an object that represents a currently running or completed asynchronous operation.

Methods

Abort

This method requests an abort of the asynchronous operation. It raises an error if an operation cannot be aborted.

Properties

CanAbort (Get)

This property returns True if an asynchronous operation can be aborted.

Completed (Get)

This property returns True if the asynchronous operation completed, irrespective of whether it completed successfully, was aborted, or was erroneous.

Error (Get)

This property returns an IAsyncError object that represents the error that occurred in the asynchronous operation. NULL is returned if there was no error.

HasProgressInfo (Get)

This property is True if this asynchronous operation can provide progress information.

HasResult (Get)

This property is True if the operation will provide a result. The resulting data can be obtained using the Result property, but only after the operation completed.

Owner (Get)

This property returns the object on which the asynchronous operation is being performed on.

Progress (Get)

This property returns the current progress value.

ProgressFirst (Get)

This property returns the lower bound of the progress range or start value.

ProgressLast (Get)

This property returns the upper bound of the progress range or end value.

Result (Get)

This property returns the result data from the asynchronous operation, if data is available.

Status (Get)

This property returns the current status of the operation.

IBooleanQueryConfig Interface

This interface provides properties that affect search parameters.

Properties

CaseInsensitive (Get)

This property indicates whether word searches are case insensitive.

CaseInsensitive (Put)

When the CaseInsensitive property is True, words are case insensitive. This is particularly useful when locating acronyms.

HitStems (Get)

If True, hit words or their stems.

HitStems (Put)

If True, hit words or their stems.

HitSynonyms (Get)

If True, hit words or their synonyms.

HitSynonyms (Put)

If True, hit words or their synonyms.

SearchAnswer (Get)

If True, search Answer field of Entries.

SearchAnswer (Put)

If True, search Answer field of Entries.

SearchKeywords (Get)

If True, search Keywords defined for Entries.

SearchKeywords (Put)

If True, search Keywords defined for Entries.

SearchQuestion (Get)

If True, search Question field of Entries.

SearchQuestion (Put)

If True, search Question field of Entries.

IDeletedEntryData Interface

Interface of an object that contains the data of a logically deleted entry.

Methods

CreateNewEntry

The CreateNewEntry method creates a new Entry and populates it with all the data (except EntryId) in this Entry.

Properties

ActivationDate (Get)

The ActivationDate property returns the date and time (in UTC format) when this Entry is activated, or NULL if there is no activation date.

AnswerPlain (Get)

The AnswerPlain property returns the Answer in plain text, with all tags removed.

AnswerXML (Get)

The AnswerXML property returns the Answer text in XML format.

Attributes (Get)

Collection of collection of attributes of this Entry

Author (Get)

The Author property returns the name of the person who authored this Entry.

AuthorEMail (Get)

The AuthorEmail property returns the e-Mail address of the author of this Entry.

Created (Get)

The Created property returns the date and time when this Entry was created, in UTC format.

EntryId (Get)

The EntryID property returns the Entry Id of this entry.

ExpirationDate (Get)

The ExpirationDate property returns the date and time when entry is deactivated, in UTC format. A NULL value is returned if there is no expiration date.

FAQ (Get)

The FAQ property returns the IFAQ object that owns this Entry.

IsActive (Get)

Returns True if Entry is active. NOTE: As the entry is logically deleted, 'Active' here means just that the flag is set and if the entry were to be undeleted, it would become active.

Keywords (Get)

The keywords property returns a collection of keywords of this Entry.

LastModified (Get)

The LastModified property returns the date and time when this Entry was last changed, in UTC format.

NotifyAddresses (Get)

The NotifyAddresses property returns a collection of e-mail addresses to be notified when this entry is a hit.

QuestionPlain (Get)

The QuestionPlain property returns the Question in plain text format, with all tags removed.

QuestionXML (Get)

The QuestionXML property returns the Question text in XML format.

UseInResponse (Get)

The UseInResponse property is True if this Entry should be included when generating a response.

UserData (Get)

The UserData property returns user-defined data associated with this Entry. An e-FAQ user is someone who sends queries to a response mailbox.

Version (Get)

The Version property returns the version number of the Entry. This number is incremented each time the Entry is changed.)

IDeletedFAQData Interface

Interface of an object that contains the configuration data of a logically deleted FAQ.

Properties

Created (Get)

The FaqID property returns the UTC date and time when this FAQ was created.

Description (Get)

Description text of FAQ

EMailAuthoring (Get)

Allow/disallow e-mail based authoring of entries in this FAQ

EntryAttributes (Get)

The EntryAttributes property returns a collection of IEntryAttributes for this FAQ.

FaqId (Get)

The FaqId property returns the name of the FAQ.

Killwords (Get)

Collection of killwords of this FAQ

LanguageId (Get)

The LanguageID property returns a numeric langauage code that identifies the language for this FAQ. Currently, only 1033 (English US) and 0 (language neutral) are supported.

LastModified (Get)

The date and time when this FAQ configuration was last changed (in UTC).

Name (Get)

Name of the FAQ

Password (Get)

Password of the FAQ for e-mail authoring

Server (Get)

The Server property returns the server object that owns this FAQ.

TokenizeNumbers (Get)

The TokenizeNumbers property indicates whether tokenization of numbers in entries of this FAQ is enabled or disabled.

UserData (Get)

The UserData property returns user-defined data in XML format.

WebConfigData (Get)

The WebConfigData property returns XML configuration data used by web applications, such as the e-FAQ Knowledge Manager.

IEntries Interface

This interface manipulates a collection of entry objects. You can return the number of attributes in the collection, retrieve or remove individual entries in the collection.

Methods

Add

This method adds an entry object to a collection of entry objects in the FAQ. The entry will be sent to the server to add to this FAQ. This operation may take a few seconds.

Item

The Item method returns the entry object found at the given 1-based index of a collection, or based upon the string value of the entry itself.

Remove

This method logically deletes an entry. To specify which Entry to remove, pass the a 1-based index number, the identifier of the entry as a string, or an IEntry object.

Properties

Count (Get)

This read-only property returns the number of elements in a collection of entry objects.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of entry values.

IEntries2 Interface

This interface refines the IEntries interface and adds various new methods.

Methods

Refresh

This method performs a shallow refresh of the list of Entries. A shallow refresh means that just the list is refreshed. The entries in the list are not refreshed or invalidated. Use the IEntry::Revert method to force a refresh of the individual entries.

RemoveEx

This method removes the Entry from the server. You can specify whether to immediately purge the Entry. Please refer to the IEntries::Remove method for more information.

IEntry Interface

This interface allows you to manipulate data items associated with an entry object.

Methods

ApplyChanges

This method sends local changes made to entry data to the e-FAQ server. This method fails if the entry data is not actually dirty.

Clone

This method creates a new entry object that has the same data as the current entry. The new object will get a new Entry ID. The new entry is not attached to a FAQ.

LogAsUsed

This method logs a use of this entry for reporting purposes. Optionally, LogAsUsed accepts QueryResultSet, QueryResult (both IDispatch), or QueryId (VT_I4) as argument to associate entry usage with a query.

Revert

This method fetches stored data from the server, discarding any local changes that you might have made to the entry object.

Properties

AnswerPlain (Get)

The AnswerPlain property returns the answer to a FAQ in plain text format. All tags are stripped, entities are substituted and
 tags are converted to newlines (\n).

AnswerPlain (Put)

The AnswerPlain property assigns plain text to the answer to a FAQ.

AnswerXML (Get)

This property returns the answer to a FAQ in XML format.

AnswerXML (Put)

This property assigns an XML-formatted string to the answer to a FAQ.

Attribute (Get)

This property allows you to read the attribute of an entry. An attribute is a tag that denotes an entry as belonging to a subset of entries.

Attribute (Put)

This property writes the attribute of an entry.

Author (Get)

This property returns the name of the person (author) who created this entry.

Author (Put)

This property assigns the name of the person who created the entry.

AuthorEMail (Get)

This property returns the E-Mail address of the author of an e-FAQ entry.

AuthorEMail (Put)

This property assigns the E-Mail address of an author to an e-FAQ entry.

Created (Get)

This property returns the date and time when an entry was first added to the FAQ.

Created (Put)

You can assign a value to this property if the entry is a new entry (not yet attached to a FAQ). You should only set the Created time when you also set an EntryId; for example, to copy entries from one server to another.

EntryId (Get)

This property returns the Entry ID of an entry.

EntryId (Put)

This property assigns as string value to the Entry ID. You can assign a value only if the entry is a new entry, and the entry is not yet attached to a FAQ.

FAQ (Get)

This read-only property returns a FAQ object of the FAQ that owns this entry. NULL is returned if the entry is new and has not been added to a FAQ).

IsActive (Get)

You can use the IsActive property to determine whether this Entry is active or inactive.

IsActive (Put)

You can use the IsActive property to determine whether this Entry is active or inactive.

IsDirty (Get)

This read-only property is True if the entry has been modified locally and changes have not been sent to the server yet (using the IEntry::ApplyChanges method).

IsNewEntry (Get)

This read-only property returns True if the entry is a new (or cloned) entry object that has not yet been added to a FAQ.

Keywords (Get)

This property allows you to access the collection of keywords for the entry.

Keywords (Put)

This property allows you to modify the collection of keywords for the entry.

LastModified (Get)

This read-only property returns the time when the entry was last modified.

NotifyAddresses (Get)

This property allows you to access a collection of e-mail addresses to notify when entry is hit in an e-mail query.

NotifyAddresses (Put)

This property allows you to modify a collection of e-mail addresses to notify when entry is hit in an e-mail query.

QuestionPlain (Get)

The QuestionPlain property returns the question from a FAQ in plain text format. All tags are stripped, entities are substituted and
 tags are converted to newlines (\n).

QuestionPlain (Put)

The QuestionPlain property assigns a question (in plain text format) to a FAQ.

QuestionXML (Get)

This property returns a question in XML format.

QuestionXML (Put)

This property allows you to assign XML-formatted text to a question.

Server (Get)

This read-only property returns the server object which the Entry belongs to, or NULL if the entry is not a member of a FAQ.

UseInResponse (Get)

This property indicates whether the entry will be included in a response e-mail message.

UseInResponse (Put)

This property determines whether the entry will be included in a response e-mail message.

UserData (Get)

The UserData of an Entry can be used to store any user defined data with the entry. This data is not interpreted by e-FAQ and is just stored with the entry. This data can be up to 1024 characters long and must constitute well-formed XML.

UserData (Put)

The UserData that you assign to an Entry must be in XML format. Your XML statements can be up to 1024 characters in length. You don't have to specify a root element, since e-FAQ implies <USERDATA> your data </USERDATA>.

Version (Get)

This property returns the version number of the entry. Version numbers are incremented each time that an entry is modified. If this value is 1, the entry has not been modified since it was created.

Version (Put)

If the entry object is new (has not been attached to a FAQ) you can assign a value to this property. This may be used to restore backups, etc.

IEntry2 Interface

This interface refines the IEntry interface and adds various new methods to support new features such as feedback, activation/expiration dates, and locking.

Methods

CopyFrom

The CopyFrom method copies data from the specified entry, to this entry. Only the fields that can be written are copied. Note: Feedback is not copied!

Lock

This method locks the entry on the Server to prevent changes by other clients while the entry is being changed. If the lock succeeds, the current state of the entry is retrieved from the server to ensure the client holds the most current data.

Unlock

This method unlocks the entry if it is locked. An exception is thrown if the entry is not locked by this client.

Properties

ActivationDate (Get)

The ActivationDate property returns the UTC date and time when the entry is activated, or NULL if the entry does not have an activation date. See also the IEntry::IsActive property.

ActivationDate (Put)

Sets the UTC date and time when entry is activated (NULL: no activation date). Please refer to the IEntry::IsActive property for more details.

Attributes (Get)

The Attributes property returns a collection of attributes of this Entry.

Attributes (Put)

Assigns the contents of a collection of attributes to the attributes collection of this entry. The attributes are identified by name and it is thus not necessary that the source collection is from an entry of the same FAQ.

ExpirationDate (Get)

The ExpirationDate property returns the UTC date and time when the entry will be deactivated. If NULL, the entry has no expiration date and thus never expires. Please refer to the IEntry::IsActive property for details.

ExpirationDate (Put)

Assigns the date and time (in UTC format) when this entry will be deactivated. If you set the date and time to NULL, the entry won't have an expiration date and will never expire. Please refer to the IEntry::IsActive property for details.

Feedback (Get)

The Feedback property returns a collection of feedback items of this Entry.

IsLocked (Get)

The IsLocked property indicates whether this entry is locked by this client.

IEntryAttribute Interface

This interface allows you to manipulate data items associated with an entry attribute, such as the attribute name, attribute ID, and the FAQ that owns the attribute.

Properties

AttributeId (Get)

This property returns the attribute's identifier, or -1 if the attribute is not attached to a FAQ.

FAQ (Get)

This property returns the FAQ object that owns the attribute. If the attribute is not owned by a FAQ, NULL is returned.

Name (Get)

This property returns the name of the attribute.

Name (Put)

You can also change the name of an attribute.

IEntryAttributes Interface

The IEntryAttributes interface manipulates collections of Entry attribute objects. You can return the number of attributes in the collection, retrieve the attribute at a given index, add or remove attributes from the collection.

Methods

Add

This method adds an entry attribute to the collection. The attribute can be specified by name or passed as an object to the collection. An IEntryAttribute object is returned.

Clear

The Clear method removes all attributes from a collection.

Item

The Item method returns the entry attribute at the given 1-based index of a collection. The attribute can optionally be specified by name.

Remove

This method removes an attribute from a collection of entry attributes. To specify which item is removed, the function is passed an index number (1-based), the string value of the attribute, or an object.

Properties

Count (Get)

This read-only property returns the number of elements in a collection of entry attribute objects.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of entry attribute values.

IEntryFeedback Interface

This interface refines the IEntry interface and adds various new methods to support new features such as feedback, activation/expiration dates, and locking.

Methods

Add

The Add method adds a new feedback item to the Entry. The feedback item must be created through IServer2::CreateNewEntryFeedbackItem method and initialized before adding to the entry.

Clear

The Clear method removes all feedback items from the Entry.

Find

The Find method retrieves the feedback item with the given identifier. Null is returned if the item is not found.

Item

This method retrieves the feedback item at the specified index (1 based). If the specified index is out of range, an exception is thrown.

Refresh

Refreshes the list of Feedback items from the Server

Remove

The Remove method removes a feedback item from the Entry (undoes the feedback). To specify which item to remove, specify its index number or pass the feedback object to this function.

Properties

Count (Get)

The Count property returns the total number of feedback items in the collection.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate items in a collection of feedback items.

IEntryFeedbackItem Interface

An interface of objects containing confidence and alternate wording feedback for an Entry.

Methods

ApplyChanges

The ApplyChanges method send changes made locally to the Feedback item to the e-FAQ Server.

Clone

The Clone method creates a new Entry Feedback Item with the same data as this one. The new feedback item is not added to an Entry yet.

Revert

The Revert method fetches current data from the server, discarding any local changes to this object.

Properties

AlternateWording (Get)

This property returns alternate question wording for this Entry as plain text (not XML).

AlternateWording (Put)

The AlternateWording property can be used to assign alternate question wording for this Entry. Your copy must be supplied as plain text, rather than XML.

Author (Get)

The Author property returns the name of the author of the feedback item.

Author (Put)

The Author property assigns a new name as the author of the feedback item.

Comment (Get)

The Comment property returns a custom comment stored with the feedback.

Comment (Put)

The Comment property can be used to change a custom comment stored with feedback.

Created (Get)

The Created property returns the UTC date and time when the feedback item was created.

Created (Put)

The Created property changes the creation date and creation time entry for this feedback item.

Entry (Get)

The Entry property returns the Entry which owns this feedback item.

ExpandSynonyms (Get)

The ExpandSynonyms property indicates whether to expand the synonyms of words in the alternate wording.

ExpandSynonyms (Put)

The ExpandSynonyms property determines whether to expand the synonyms of words in the alternate wording. This property cannot be changed if feedback is attached to the entry.

FeedbackId (Get)

The FeedbackId property return a long number that identifies this feedback item.

IsDirty (Get)

The IsDirty property returns True if this Feedback item has been changed locally.

IsNewItem (Get)

The IsNewItem property indicates whether this Feedback item has been added to an Entry. True is returned if the item is new and has not been added to an Entry.

LastModified (Get)

The LastModified property returns the date and time when this feedback item was last modified. The date is returned in UTC format.

StrengthMeasure (Get)

The StrengthMeasure property returns the positive or negative strength measure that was applied to this feedback item. The value range depends on the type of feedback (alternate wording or entry confidence).

StrengthMeasure (Put)

You can assign positive or negative strength measure to this feedback item. The range of acceptable values depends on the type of feedback (alternate wording or entry confidence).

IEntryRecycleBin Interface

IEntryRecycleBin objects are collections of entries that have been logically deleted from a FAQ.

Methods

GetEntryData

The GetEntryData method returns the data of the logically deleted Entry. The returned object only provides access to a limited part of the data.

GetEntryIds

The GetEntryIds method returns a list of EntryIDs of the Entries that are logically deleted.

IsDeleted

The IsDeleted method returns True if the FAQ has an Entry with the given ID that has been logically deleted.

Purge

The Purge method physically deletes the specified Entry, if it has been marked as deleted.

Undelete

The Undelete method Un-deletes the Entry with the given ID, if it has been logically deleted but is physically still in the database.

IFAQ Interface

This interface allows you to manipulate objects representing a FAQ.

Methods

ApplyChanges

This method sends changes made to the FAQ to the e-FAQ server. The method returns True if the FAQ needs to be reindexed. This method fails if the data of the FAQ has not changed. See also IFAQ::IsDirty.

Clone

This method creates a new FAQ object and initializes the new FAQ with the same data as the object cloned. Entries and Aliases are not copied. Killwords and EntryAttributes are copied. The clone object is not attached to an e-FAQ server.

GetEntry

This method retrieves an entry object by specifying its Entry ID. The EntryID must belong to an entry in this FAQ. NULL is returned if the entry object is not found.

Reindex

Re-indexes the FAQ. WARNING: Takes potentially very long time! Consider using ReindexAsync

ReindexAsync

This method reindexes a FAQ asynchronously. The FAQ remains available while indexing takes place.

Revert

This method abandons local changes made to a FAQ configuration and fetches current data from the server. Local changes are discarded.

Properties

Aliases (Get)

This read-only property returns a collection of aliases of this FAQ.

Created (Get)

This property returns the date and time when the FAQ was added to the e-FAQ server.

Created (Put)

If the object is a new FAQ, you can assign the date and time to this property.

Description (Get)

This property returns comment text stored with the FAQ.

Description (Put)

This property allows you to set descriptive text stored with the FAQ.

EMailAuthoring (Get)

This boolean property indicates whether e-mail based authoring of entries in this FAQ is allowed or disallowed.

EMailAuthoring (Put)

This boolean property allows you to allow or disallow e-mail based authoring of entries in this FAQ.

Entries (Get)

This read-only property returns a collection of entries in the FAQ.

EntryAttributes (Get)

This property returns a collection of entry attributes defined for this FAQ.

EntryAttributes (Put)

This property assigns a collection of entry attributes defined for this FAQ.

FaqId (Get)

This read-only property returns the numeric FAQ ID (database row id).

IsDirty (Get)

The IsDirty property is True if a FAQ configuration has been changed locally, and changes have not been applied to the server yet. Adding or removing entries does not change the FAQ configuration and thus does not mark a FAQ as dirty.

IsNewFAQ (Get)

This read-only property is True if the FAQ is a new FAQ object that has not yet been added to a Server.

Killwords (Get)

This property allows you to read the collection of killwords of this FAQ.

Killwords (Put)

This property allows you to copy the killword collection of one FAQ to another FAQ.

LastModified (Get)

This read-only property returns the date and time when the FAQ configuration was last changed.

Name (Get)

This property returns the name of this FAQ.

Name (Put)

If the object is a new FAQ, you can assign the name of this FAQ.

NeedsReindexing (Get)

This property returns True if changes to the FAQ configuration require the FAQ to be re-indexed. This property directly queries the current state of the FAQ on the server and does not use cached information.

Password (Get)

This property reads the password of the FAQ for e-mail authoring.

Password (Put)

This property writes the password of the FAQ for e-mail authoring.

Server (Get)

The Server property returns the server object of the FAQ, or NULL if the object is not yet a member of a server.

TokenizeNumbers (Get)

This property allows you to enable or disable tokenization of numbers in entries of the current FAQ.

TokenizeNumbers (Put)

This property indicates whether tokenization of numbers in entries of the current FAQ is enabled or disabled.

UserData (Get)

This property returns a user-defined string stored with the FAQ. The data must be well formed XML.

UserData (Put)

This generic property allows you to store up to 1024 characters of user defined data with this FAQ. The data must constitute well formed XML. You don't have to specify a root element, since e-FAQ implies <USERDATA> your data </USERDATA>.

IFAQ2 Interface

Collection of logically deleted entries.

Methods

Lock

Locks the FAQ on the Server to prevent changes to the FAQ configuration from other clients. If the lock succeeds, the current state of the FAQ configuration is retrieved from the server to ensure the client holds the most current data.

Unlock

The Unlock method unlocks the FAQ if it is locked. If the FAQ is not locked by this client, an exception is thrown.

Properties

EntryRecycleBin (Get)

The EntryRecycleBin property returns an IEntryRecycleBin object that provides access to logically deleted Entries.

IsLocked (Get)

The IsLocked property indicates whether the FAQ is locked by this client.

LanguageId (Get)

The LanguageId property returns a number that indicates the language used in the FAQ. Currently, only two codes are supported: 1033 (English US) and 0 (language neutral).

LanguageId (Put)

You can assign a language to the FAQ by passing a numeric language code. Currently, only two codes are supported: 1033 (English US) and 0 (language neutral).

WebConfigData (Get)

The WebConfigData property returns web configuration in XML format for web applications such as the e-FAQ Knowledge Manager.

WebConfigData (Put)

You can update the configuration data used by web applications (such as the e-FAQ Knowledge Manager) by passing well-formed XML to this function.

IFAQRecycleBin Interface

This interface creates an object that is a collection of logically deleted FAQ entries.

Methods

GetFAQData

The GetFAQData method returns the data of the logically deleted FAQ. The returned object only provides access to a limited part of the FAQ data.

GetFAQNames

The GetFAQNames method returns a list of names of the FAQs that are logically deleted.

IsDeleted

The IsDeleted method returns True if a FAQ with the given name has been logically deleted.

Purge

The Purge method physically deletes the given FAQ, if it has been marked as deleted.

Undelete

This method un-deletes the FAQ with the given name, if it has been logically deleted but is physically still in the database.

IFaqReindexCompletionCallback Interface

IFaqReindexCompletionCallback is a callback interface that is called when FAQ indexing is completed.

Methods

IndexingCompleted

This method is called during re-indexing of a FAQ to report progress.

IFaqReindexProgressCallback Interface

IFaqReindexProgressCallback is a callback interface called during re-indexing of a FAQ to report progress.

Methods

IndexingCheckPoint

This method is called during re-indexing of a FAQ to report progress.

IFAQs Interface

This interface provides access to collections of FAQs (frequently asked questions.)

Methods

Add

This method adds a FAQ to the e-FAQ Server.

Item

This method returns the IFAQ object found at the given 1-based index of a collection, or based upon the name of the entry itself.

Remove

This method logically deletes a FAQ. To specify which FAQ to remove, pass the a 1-based index number, the name of the FAQ as a string, or an IFAQ object.

Properties

Count (Get)

This read-only property returns the number of FAQs in a collection of FAQ objects.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of FAQ values.

IFAQs2 Interface

This interface refines the IFAQs interface and adds various new methods.

Methods

Refresh

Performs a shallow refresh of the list of FAQs. A shallow refresh here means that just the list is refreshed. The FAQs in the list are not refreshed or invalidated. Use the IFAQ::Revert method to force a refresh of the individual entries.

RemoveEx

The RemoveEx method removes the FAQ from the server. The bPurgeNow parameter allows you to specify whether to immediately purge the FAQ. Please refer to the IFAQs::Remove method for more information.

IKeyword Interface

The objects created using this interface represent keywords. You can control whether keywords are expanded, whether synonyms are added, whether Parts of Speech (POS) are written, and whether root keywords are indexed.

Properties

Adjective (Get)

The Keyword is an adjective when this property is True.

Adjective (Put)

To indicate that the keyword is an adjective, set this property to True.

Adverb (Get)

The Keyword is an adverb when this property is True.

Adverb (Put)

To indicate that the keyword is an adverb, set this property to True.

ExpandAliases (Get)

This property determines whether a keyword will be expanded against its aliases. The keyword is expanded when ExpandAliases is True.

ExpandAliases (Put)

Set this property True to expand the keyword against its aliases.

ExpandSynonyms (Get)

Set this property True to add synonyms of the keyword to the keyword index.

ExpandSynonyms (Put)

This property indicates whether synonyms of the keyword will be added to the keyword index. Synonyms are added when when ExpandSynonyms is True.

GuaranteedHit (Get)

When this property is True, the entry will be chosen, irrespective of grade, if a keyword is found.

GuaranteedHit (Put)

To force an entry to be chosen if a keyword if found, irrespective of grade, set this property to True.

Noun (Get)

The Keyword is a noun when this property is True.

Noun (Put)

To indicate that the keyword is a noun, set this property to True.

Stem (Get)

If this property is True, the root of the keyword will be added to the index.

Stem (Put)

To add the root of the keyword to the index, set this property True.

Verb (Get)

The Keyword is a verb when this property is True.

Verb (Put)

To indicate that the keyword is a verb, set this property to True.

Word (Get)

This read-only property returns the keyword that the object represents. Note that the word of a keyword cannot be changed. You have to delete the keyword and add a new one.

IKeywords Interface

The IKeywords interface allows developers to manipulate collections of keywords of an entry.

Methods

Add

This method adds a keyword to the entry and returns a keyword object.

Clear

This method removes all keywords from the entry.

Item

This method returns a keyword object from the collection. To return a specific item, you can specify a 1-based index number, or the name of the object as a string value.

Remove

This method removes a keyword from the entry. To specify the item to remove, pass the function a 1-based index, a string value that identifies the item, or a keyword object.

Properties

Count (Get)

This read-only property returns the number of keywords in the collection.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of entries in the collection.

IKillwords Interface

This interface manipulates a collection of killwords of a FAQ. KillWords are words (strings) that e-FAQ ignores when searching for an answer.

Methods

Add

This method adds a killword to the FAQ.

Clear

This method removes all killwords from the FAQ.

Item

This method returns the killword found at the given 1-based index of a collection.

Remove

This method removes a killword from the FAQs. To specify the item to remove, pass the function a 1-based index, or the killword text as a string value.

Properties

Count (Get)

This read-only property returns the number of words in a collection of killwords.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of killwords in a collection.

ILockInfo Interface

Interface of objects representing information about a FAQ or Entry locking operation.

Properties

CurrentOwner (Get)

The CurrentOwner property returns an object containing information about the current owner of a lock.

Failed (Get)

The Failed property returns True if acquiring the lock failed (e.g. the lock not forced and somebody else holds a lock on the same resource).

LockForced (Get)

The LockForced property is True if the lock was acquired by forcing it from another owner.

PreviousOwner (Get)

The PreviousOwner property returns an ILockOwnerInfo object containing information about the previous owner of the lock (if lock was forced).

Succeeded (Get)

The Succeeded property is True if a lock was successfully acquired.

ILockOwnerInfo Interface

The objects you create using this interface provide information about the owner of a FAQ or Entry lock.

Properties

ClientId (Get)

The ClientId property returns the Client Identifier of the owner of the lock.

LockMessage (Get)

The LockMessage property returns the message passed to the Lock method by the owner of the lock.

SessionId (Get)

The SessionId property returns the Session Identifier of the owner of the lock.

Timestamp (Get)

The Timestamp property returns the time when a lock had been obtained. The time is returned in UTC format.

INotifyAddresses Interface

The INotifyAddresses interface allows developers to maintain collections of e-mail addresses that e-FAQ notifies when an entry is used as a response. This can be used to notify authors that an entry has been used.

Methods

Add

This method adds an e-mail address to the collection.

Clear

This method removes all e-mail addresses from the collection.

Item

This method returns the e-mail address found at the given 1-based index of a collection.

Remove

This method removes an e-mail address from the collection. To specify the item to remove, pass the function a 1-based index, or the e-mail address as a string value.

Properties

Count (Get)

This read-only property returns the number of e-mail addresses in a collection.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of e-mail addresses in the collection.

IQuery Interface

This interface defines and executes a FAQ query. It exposes properties that affect execution and logging operations, and can duplicate query objects.

Methods

Clone

This method clones (copies) the current object, for reuse in another query.

Execute

This method executes a query object on the server. It returns a QueryResultSet object.

Properties

AbsThreshold (Get)

This property returns the absolute score threshold. Entries with an absolute score below this value are not considered. The absolute score of an entry can be between 0.0 to 1.0.

AbsThreshold (Put)

This property returns the absolute score threshold. Entries with an absolute score below this value are not considered. The absolute score of an entry can be between 0.0 to 1.0. The absolute scores of all entries is used to calculate the Grade.

AnswerWordWeight (Get)

This property returns the relative weight given to words in answer text compared to words in question text of the FAQ entries. Values must be in the range 0.0 to 1.0. The default = 0.7.

AnswerWordWeight (Put)

This property assigns the relative weight given to words in answer text compared to words in question text of the FAQ entries. Values must be in the range 0.0 to 1.0. The default = 0.7.

CacheResultEntries (Get)

This property determines whether entries selected by the query are cached on the client. When this property is True, all entries chosen by the query are cached to reduce server round trips.

CacheResultEntries (Put)

To reduce server round trips, by caching all entries chosen by the query, set this property True.

LogQuery (Get)

This property indicates whether a query is logged for reporting purposes. The default value is True, indicating that the query will be logged for reporting.

LogQuery (Put)

Set this property True to specify that the result of the query is logged for reporting purposes.

LogResultAsUsed (Get)

This property indicates whether e-FAQ automatically logs entries in the result-set as used, in order to avoid server round trip.

LogResultAsUsed (Put)

When this property is True, e-FAQ automatically logs entries in the result-set as used. This avoids server round trip.

MaxHits (Get)

The property returns the maximum number of answers in the result-set.

MaxHits (Put)

The property sets the maximum number of answers in the result-set. The default is 5.

MinGrade (Get)

This property returns the minimum grade threshold, which is a number between 1.0 and 10.0. The default value is 2.0.

MinGrade (Put)

This property assigns the minimum grade threshold, which is a number between 1.0 and 10.0. The default value is 2.0.

POSTagQuestion (Get)

When True, this property indicates that Part Of Speech tagging is applied to the question text. This will slow down queries, but the result will be better.

POSTagQuestion (Put)

To apply Part Of Speech tagging to the question text, set POSTagQuestion True. Enabling parts of speech tagging will slow down the query slightly, but will increase the match quality.

QueryFAQs (Get)

This property returns a collection of FAQs to be queried.

QueryFAQs (Put)

This property assigns a collection of FAQ names to be queried. Use this to copy the FAQs to be queried from one query object to another.

Question (Get)

This property contains the question text that will be asked.

Question (Put)

This property assigns the question text that will be processed.

RequestId (Get)

This property contains a string representing the custom request identifier to be logged into the remote query table for reporting purposes.

RequestId (Put)

This property assigns a string representing the custom request identifier to be logged into a remote query table for reporting purposes.

Server (Get)

This property returns a server object that the query will be run against.

IQuery2 Interface

This interface refines the IQuery interface and adds various methods and properties to support additional query features, such as Boolean expression queries, XML result queries, etc.

Methods

ExecuteToXML

The ExecuteToXML method executes the query and returns the query result as an XML document.

GetProperty

The GetProperty method retrieves the value of a named query property. Properties allow you to pass additional information to the server, such as hints for the matcher engine, without necessitating changes to the interface.

SetProperty

The SetProperty method sets the value of a named query property. Properties allow passing additional information to the server, such as hints for the matcher engine, without necessitating changes to the interface.

Properties

BooleanQueryConfig (Get)

The BookeanQueryConfig property returns an IBooleanQueryConfig object representing configuration data for boolean expression queries.

BooleanQueryConfig (Put)

Assigns the data of a boolean query configuration object to this query configuration object.

LogQueryResult (Get)

When the LogQueryResult property is True, the result of the query will be logged for reporting. When this property is False, only the query itself is logged, but not the actual result of the query.

LogQueryResult (Put)

You can use the LogQueryResult property to specify whether to log the result of the query for reporting. When this property is False, the query is logged, but not the actual result of the query. This saves space in the query log. The default is True.

Mode (Get)

The Mode property returns the mode of the query. Please see put_Mode for details.

Mode (Put)

You can also use the Mode property to set the desired mode of the query.

IQueryFAQ Interface

This interface provides an object representing a FAQ that will be queried during execution of a query.

Properties

EntryAttributes (Get)

This property returns a collection of entry attributes that qualifies which entries of this FAQ are to be considered in the query.

EntryAttributes (Put)

You can also assign a collection of entry attributes to qualify a query with.

FAQ (Get)

This property returns the FAQ that this object represents.

Query (Get)

This property returns the IQuery object that owns this object.

IQueryFAQs Interface

The IQueryFAQs interface provides a collection of objects representing FAQs to be queried in a query.

Methods

Add

This method adds a FAQ to the collection. You must pass the name of the FAQ or an IFAQ object as parameters. An IQueryFAQ object is returned.

Clear

This method removes all FAQs from the collection.

Item

This method returns an IQueryFAQ object. You can pass this method a 1-based index number, the name of the FAQ, or an IFAQ object.

Remove

This method removes an FAQ from the collection. To specify the item to remove, pass the method a 1-based index number, the FAQ name as a string, an IFAQ object, or an IQueryFAQ.

Properties

Count (Get)

This read-only property returns the number of FAQ names in a collection.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of FAQ names in the collection.

IQueryResult Interface

The IQueryResult interface provides statistics about an Entry selected as the result of a query. A query result object represents one row in the result set.

Methods

LogAsUsed

The LogAsUsed method logs a use of the result entry for reporting purposes. Please refer to IEntry::LogAsUsed for a discussion of this feature.

Properties

AbsoluteScore (Get)

This property returns the absolute score of the entry.

Entry (Get)

This property returns the entry object associated with the query result.

EntryId (Get)

This property returns the identifier (entry ID) of the query result. This is more efficient than using IEntry::EntryId, since the Entry potentially has to be fetched from the server. IQueryResult::EntryID is useful when only the entry ID is needed.

Grade (Get)

This read-only property returns the grade of the entry in this query result.

GuaranteedHit (Get)

True if result is because a 'GuaranteedHit' keyword appeared in the question.

Mean (Get)

This read-only property returns the mean absolute score of the entry in the query result.

QueryResultSet (Get)

This read-only property returns the query result set that the object is a member of, as an IQueryResultSet object.

StdDev (Get)

This read-only property returns the standard deviation of the absolute scores in the query result.

IQueryResults Interface

The IQueryResults interface allows you to enumerate a collection of query result objects, to obtain the total number of item in the collection, and to retrieve an individual item.

Methods

Item

This method returns a query result object from the collection. To return a specific item, you can specify a 1-based index number, or the EntryId of one of the entries in the result set.

Properties

Count (Get)

This read-only property returns the number of query result objects contained in the collection.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate instances of query result objects in a collection.

IQueryResultSet Interface

The IQueryResultSet interface provides access to query result set objects.

Methods

CacheEntries

The CacheEntries method ensures that all entries appearing in the result set are cached locally on the client. A single request will be sent to the server to fetch all entries that are not yet present on the client.

LogAsUsed

The LogAsUsed method instructs the server to log all entries in this result set as used by this query (for reporting purposes). See IEntry::LogAsUsed for details.

Properties

Query (Get)

This read-only property returns the query object used to generate this result set.

QueryId (Get)

This read-only property returns the row id of the IEFAQ_L_Queries table, or -1 if the query was not logged (Query.LogQuery = False). See also: IQuery::LogQuery property.

QueryResults (Get)

This property returns a collection of query result elements. Elements in collection are ordered by grade in descending order (highest grade first).

Server (Get)

This property returns a server object representing the server that the query ran against.

IServer Interface

IServer is the root object for accessing the e-FAQ server. Use the IServer interface to connect to an e-FAQ server, or to determine whether a connection to the server exists.

Methods

Connect

The Connect method establishes a connection with an e-FAQ server. The e-FAQ server name, and client identifier must be passed as arguments.

CreateNewAlias

The CreateNewAlias method creates a new alias object, which you can later add to a FAQ by using the IAliases::Add method.

CreateNewEntry

CreateNewEntry creates an empty entry object, which can be added to a FAQ collection using the FAQ.Entries.Add method.

CreateNewFAQ

The CreateNewFAQ method creates an empty FAQ object. To add this new object to the server, use the IFAQs::Add method.

CreateQuery

The CreateQuery method creates a new query object.

GetEntry

The method returns the entry object specified by the EntryID (name) you supply as a string. A NULL entry object is returned if the specified EntryID does not exist.

GetFAQ

The method returns the FAQ object specified by the name or identifier that you supply as an argument. A NULL object is returned if the name or identifier does not exist.

GetServerInfo

This method returns the number of licenses, version and other information about the e-FAQ server.

Properties

ClientId (Get)

This read-only property returns the client identifier passed to the e-FAQ server using the IServer::Connect method. The client id can be used to identify the client in logs. The client id can be any string up to 64 characters long.

FAQs (Get)

The FAQs property returns a collection of FAQs from the e-FAQ server as an IFAQs object.

IsConnected (Get)

IsConnected is a read-only property that returns True if a connection has been established with an e-FAQ server. If no connection is active, the property returns False.

Name (Get)

The Name property is read-only. It returns the name of the connected e-FAQ server.

ServerNotifications (Get)

This object registers for notification callbacks.

IServer2 Interface

This interface refines the IServer interface and adds various new features, such as properties, extension functions, entry feedback support, etc.

Methods

CreateNewEntryFeedbackItem

The CreateNewEntryFeedbackItem method creates a new feedback item. You can add the new feedback item to an Entry using the IEntryFeedback collection, which is accessible via the IEntry2::Feedback property.

ExtensionFunction

The ExtensionFunction method invokes a named extension function. The first parameter is a BSTR (string value) that specifies the name of the function. Four additional, optional, VARIANT parameters may be specified.

GetProperty

The GetProperty method retrieves the value of a named Server property.

SetProperty

The SetProperty method sets the value of a named Server property.

Properties

FAQRecycleBin (Get)

Object providing access to logically deleted FAQs.

IServerNotifications Interface

The IServerNotifications interface is currently unused.

IStringList Interface

Interface for a read-only collection of strings.

Methods

Item

This method returns the string at the specified index in the collection.

Properties

Count (Get)

The Count property returns the total number of strings in the collection.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate through items in this collection.

IVariantList Interface

Interface for a collection of VARIANTs.

Methods

Item

The Item method retrieves a VARIANT from the collection based on the 1-based index number that you supply.

Properties

Count (Get)

The Count property returns the total number of items in the collection.

_NewEnum (Get)

This property returns an IEnumVARIANT interface so that you can enumerate through items in this collection.

Sample e-Faq Programs

e-Faq ships with several sample applications for those who learn by looking at source code.

FAQ Excel-Loader

This Visual Basic 6 program demonstrates how to bulk-load FAQs from an Excel file.

Visual Basic API Tester

This Visual Basic 6 application was used to test the COM API. It uses almost all of the interfaces, properties, and methods that are described in this document. You can browse, view and edit entries in a FAQ, and add new entries. You can perform queries and manage FAQs. This application was not intended to solve any particular real-world problem. Instead, it serves as a test bed and source of sample code for developers.

Query/Response Web Pages

This query/response application was developed using Active Server Pages (ASP). It demonstrates how the e-FAQ API can be used to implement e-FAQ on a web site. Users who type questions into a web-based form receive a response immediately from e-FAQ.

Excel exporter

This Microsoft Excel macro exports data from an e-FAQ server to an Excel table.

Word exporter

This Microsoft Word macro exports data from an e-FAQ server into word documents. It can be used to generate hard copies of the individual entries in a FAQ.

XML Editor

This Visual Basic application views and edits the contents of the XML files generated by the e-FAQ Bulk Loader utility.

e-FAQ Knowledge Manager

e-FAQ Knowledge Manager demonstrates how to add, delete, and update records using a web browser. These HTML pages can be used to maintain FAQ databases without using email.

Since the e-FAQ Knowledge Manager is an application written in ASP, it is shipped in source code. Consult it for information on how to use the e-FAQ COM API in a real world application. This source code is provided as-is. Any modifications or customizations that you perform are your responsibility to support. Our support of the e-FAQ Knowledge Manager is limited to installation and setup of the original code that ships with the product.

The Interaction Campaign COM API

Interaction Dialer containers in Interaction Administrator provide administrators with the *complete* functionality needed to define, administer, and automate campaigns and workflows. However, in some circumstances it is desirable to limit the level of administrative control that personnel can exert over Interaction Dialer. For example, a night supervisor who needs to start or stop campaigns does not necessarily need the ability to define workflows, or modify schedule settings.

The *Interaction Campaign COM API* empowers developers to create applications or web pages that provide appropriate supervisory control over the Dialer. Interaction Campaign COM API is included with Interaction Dialer, a companion product for the CIC platform that is sold separately from IC.

IICWorkflow Interface

Automation Interface to Workflows.

Methods

Delete

Deletes the current workflow. This operation is equivalent to deleting a workflow in Interaction Adminstrator.

Duplicate

Duplicates the current workflow to create a new workflow object. A workflow is an encapsulation of campaigns and runtime properties that define how the associated campaigns will run.

GetActiveCampaignID

Returns the active campaign's ID number.

GetCampaignIDs

Returns a list of Campaign IDs that belong to this workflow.

GetName

Returns the name of this workflow object.

GetProperties

Returns three matching arrays of workflow property names and values.

GetProperty

Returns the value of a workflow property that was specified using its internal name.

GetWorkflowID

Retrieves the WorkflowID of this workflow object.

PutProperty

Assigns a value to a workflow property specified using its internal name.

RecycleCallList

Recycles the active campaign's contact list to restart the call selection process at the beginning of the Contact List. This happens automatically after all records are processed.

ReloadZones

Call this method if you have updated a Zone Set in Interaction Administrator and wish to reload time zone information immediately, without waiting for the workflow to be stopped, restarted or transitioned.

RestartCampaign

Restarts the active campaign in this workflow.

RestartWorkflow

Restarts this workflow.

StatusAuto

Sets the workflow status to Auto, so that the workflow runs in accordance with a schedule.

StatusManualOff

Sets workflow status to 'Manual Off'. This effectively stops execution of the workflow.

StatusManualOn

Sets the workflow status to 'Manual On', which places the workflow in a running state, but not under control of a schedule.

StatusPause

Temporarily halts execution of the workflow, until you change it to an active status condition. If you call IICWorkflow::StatusPause a second time, the workflow is unpaused.

IICCampaign Interface

This interface provides an automation interface to Campaign objects.

Methods

Delete

Deletes this campaign, unless the campaign is referenced in a workflow. Campaigns that are referenced by one or more workflows cannot be deleted until all references are removed using the Workflows container in Interaction Administrator.

Duplicate

Duplicates a specified campaign to create a new Campaign object. A campaign can be considered to be a physical encapsulation of the properties and data associated with a contact list.

GetCampaignID

Retrieves the CampaignID (GUID) of this campaign object.

GetName

Retrieves the name of the current campaign.

GetProperties

Returns matching arrays of campaign property names and values.

GetProperty

Returns the value of the specified campaign property.

PutProperty

Assigns a value to the specified campaign property. The Status and Name properties cannot be changed using this method.

TestCampaign

Tests a campaign filter by appending sort and filter criteria to create an SQL select statement that returns a count of rows selected by the filter. An error message is returned if the filter does not succeed.

IICWorkflowManager Interface

Automation Interface to Workflow Manager.

Methods

GetActiveWorkflowIDs

Returns an array of all active Workflow IDs (Active=On or Manual On). Workflows that are not in an active status condition are not selected.

GetCampaign

Returns Campaign Object with Campaign ID pCID

GetCampaignIDs

This method returns a Campaign object specified by Campaign ID. Use the IICCampaign object to manipulate the campaign properties.

GetDefaultObjectID

Returns the Default Object ID for the specified object type

GetNameByID

This method returns the name property of an object specified by its object Id number. This is the name specified in Interaction Administrator when the campaign, workflow, or other type of object was defined.

GetObjectIDByName

This method returns an Object ID for the specified name and object type.

GetObjectIDsByType

This method returns a safearray of object IDs of the specified object type.

GetWorkflow

This method returns a Workflow object specified by Workflow ID. Use the IICWorkflow object to manipulate the workflow properties.

GetWorkflowIDs

Returns a safe array of all Workflow IDs, including the IDs of workflows that are not in an active status.

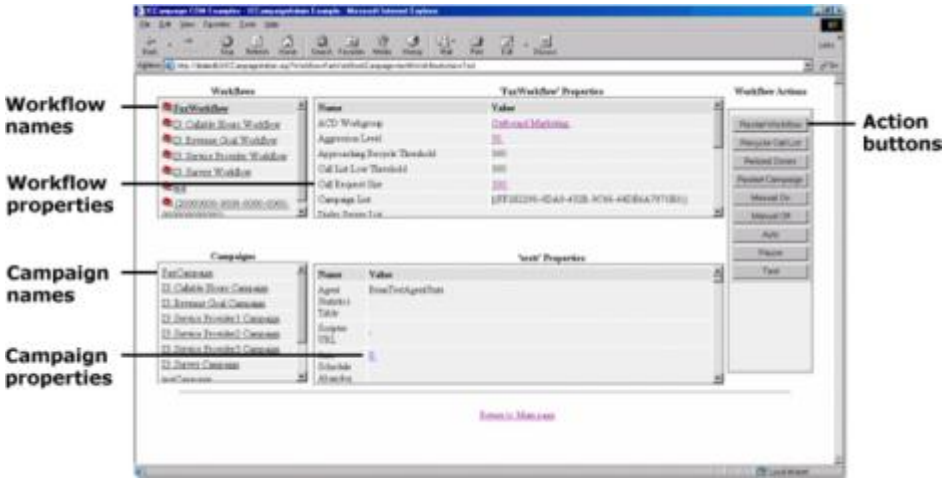
StopAllWorkflows

This method stops all running Workflows.

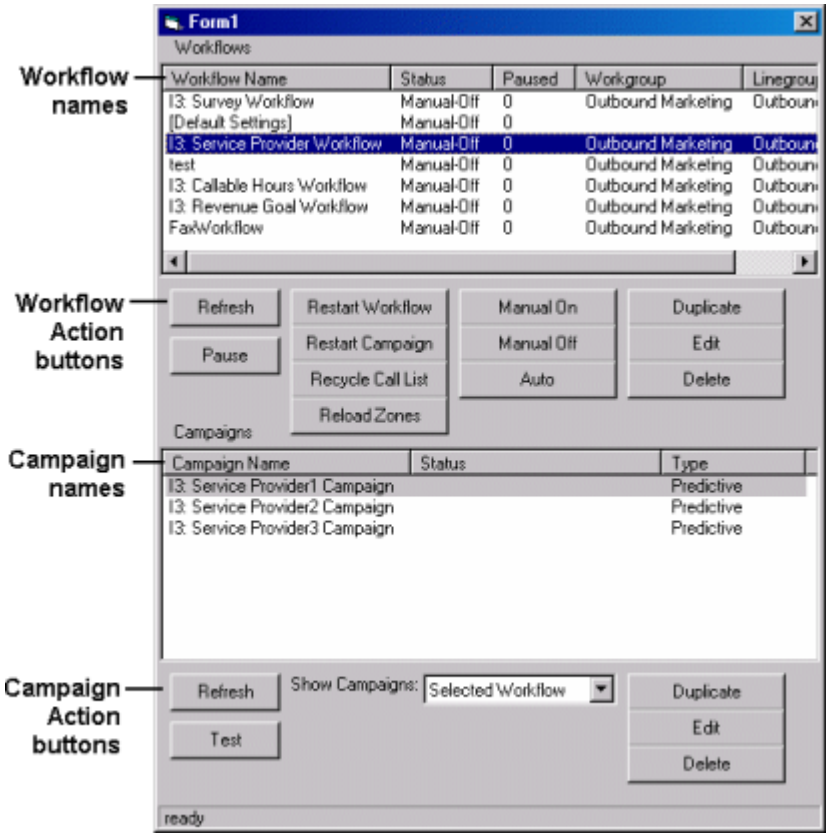
Sample Programs

When Interaction Dialer is installed, sample applications written in VB6 and ASP are placed in an examples folder on the Central Campaign server.

1. An *Active Server Pages* example demonstrates how the API can be used to expose a limited level of control over the Dialer. It allows users to view and change the runtime properties of workflows and campaigns using a web browser, rather than Interaction Administrator.



2. A *Visual Basic* example demonstrate how the Interaction Campaign COM API can be used in a Windows application. This example is functionally similar to the ASP example; it allows you to edit properties of a campaign or workflow, or to apply controlling actions.



Predictive Dialer COM API

The Predictive Dialer COM API enables programmers to develop applications that are functionally similar to the Interaction Scripter client. It provides a *callback object* that notifies an application when predictive events occur, and a *server object* whose methods manipulate and disposition data pops. The server object logs in the agent, transitions to the next stage, and updates contact list data. It provides overall control over interactions between an agent and the Outbound Dialer server.

Interaction Scripter client does not need to be running when you run a Predictive Dialer COM application. However, the Scripter client must be installed on the PC that runs your third-party application, so that component files can be shared.

IEICClientCallback2 Interface

Implement the IEICClientCallback2 object in your client application to monitor for predictive events. It provides notification when a workflow starts, stops, or transitions to a different campaign.

Methods

CampaignTransition

This method is called when a workflow transitions from one campaign to another.

DataPop

DataPop notification occurs when a new call arrives in an agent's queue from a campaign running in predictive, preview, or power mode. Call data is represented in parallel name and value arrays.

PostMessage

PostMessage is a generic method that was added for extensibility. It provides a generic callback method that the Dialer can call in future situations. This method is reserved for future use.

PreviewCallAdded

This method is called when a preview call is sent to a specific agent, before the agent actually places the call for a campaign that is running in preview mode. The data for the call is represented in parallel name and value arrays.

SetFormDefinition

This method is reserved for Interaction Scripter for Form Updates. Dialer calls this method when forms change. VB applications should implement the function in the class library as a function that returns a NULL result.

ShutdownClient

This method is called when the predictive client needs to shutdown.

WorkflowStarted

This method is called when a workflow is started.

WorkflowStopped

This method is called when a workflow is stopping.

IEICPredictiveServer2 Interface

This interface provides a server object whose methods manipulate and disposition data pops.

Methods

AssociateCall

The AssociateCall method has been deprecated. Do not use it in your application.

CallComplete

Use CallComplete when the Agent has completed a call. It sends call results, including Reason and Finish Codes, to the Outbound Dialer server.

GetFormDefinition

GetFormDefinition returns the base Interaction Scripter form definition. This method is reserved for internal use by Interactive Intelligence. Do not call this method in custom applications.

Logoff

This method logs the agent out of a workflow.

Logon

This method logs the agent into the specified workflow.

PlacePreviewCall

Places a preview call for the associated workflow. Names and Values represent contact information passed to the dialing handler.

PostMessage

Posts a message to the client callback handler. This generic method is included for future extensibility.

SetAgentType

SetAgentType sets the AgentType value for the current agent. This method is primarily used in conjunction with Finishing Agents (agents that do not receive regular campaign calls).

SetAvailability

SetAvailability sets the agent break status. If p_IsAvailable is False, then set the agent into a break state. If True, then set the agent as available to receive campaign calls.

SetMarshalledClientCallbackHandler

This method associates the Predictive Dialer COM object with an IEICClientCallback2 watcher, so that you can monitor for changes to the Predictive Dialer COM object. This watcher object must be implemented in a client application.

SetThreadedClientCallbackHandler

This method associates the Predictive Dialer COM object with an IEICClientCallback2 watcher. This allows you to monitor for changes to the Predictive Dialer COM object. Your client application must implement this watcher object.

StartReceivingCalls

The StartReceivingCalls method informs the Dialer that the associated agent is ready to start receiving campaign calls. This method is usually called after the login initialization is complete.

UpdateCallData

The UpdateCallData method updates the data associated with the current outbound contact. This method is usually called right before a contact is transferred to another party.

UpdateStage

Call the UpdateStage method when the agent has transitioned to a new stage in the call.

ValidateVersion

Interface version number being used by the Interaction Scripter predictive client. See IFREV TypeDef for values.

Properties

AvailableWorkflows (Get)

The AvailableWorkflows property returns the list of available workflows in a SAFEARRAY. The workflows in the list only include those that agents can log into. No agentless workflows are listed.

UserId (Get)

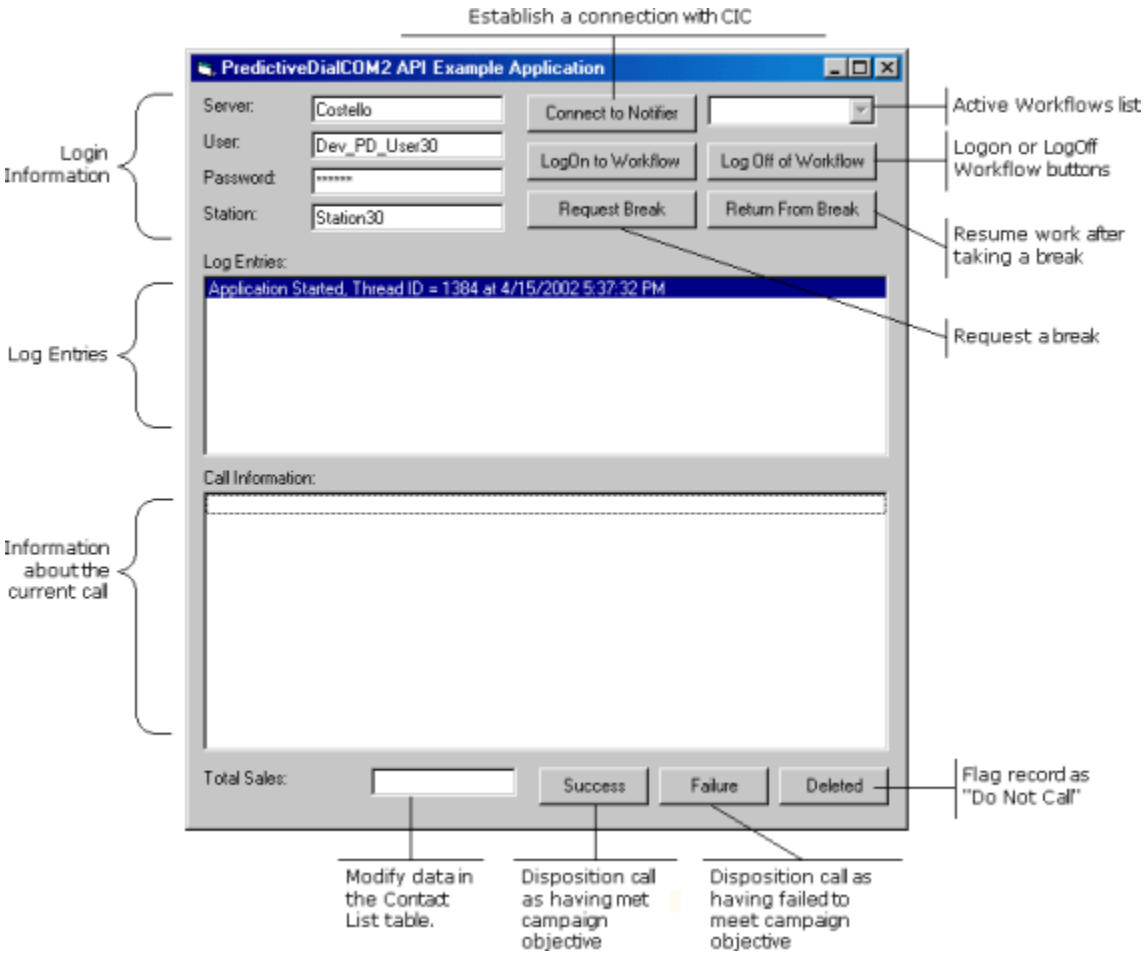
Returns the User Id of the logged in agent.

UserId (Put)

Sets the User Id of the logged in agent.

Sample Application

The API includes a sample application, written in VB6, that illustrates techniques used to create a Predictive Dialer COM application.



The Predictive Dial COM example application (PredictiveDialCOM2Example.exe)

The sample application prompts for information it needs to establish a Notifier connection with the CIC server. Once this connection is established, it populates a list of active workflows.

Once a workflow is selected, the user can join (logon) the workflow to receive calls. Information about the current call is displayed in the *Call Information* list. The user can disposition a telephone call by pressing the *Success*, *Failure*, or *Deleted* buttons.

Success indicates that the call met the overall objective of the campaign. *Failure* indicates that it did not. *Deleted* marks the current Contact List record so that the contact is never called again. The *Total Sales* field collects data that will be written to the Contact List database once the call is dispositioned. Interactions with the program are displayed as Log Entries.

Once you understand the programming behind these simple tasks, you will be able to design more advanced API applications.

Interaction Scripter API

The term "Interaction Scripter" refers to the **client application** that executes scripts, and also to the programming **API** that is used to create scripts.

Interaction Scripter client is a desktop application that executes campaign scripts written in HTML and Javascript. This application is used primarily by call center agents. Scripter's primary job is to execute *campaign scripts*—HTML pages that guide call center agents through stages of a campaign call.

Scripts are displayed inside an embedded instance of Microsoft's Internet Explorer web browser. Since the browser component is built-in, Interaction Scripter can intercept and respond to web events. It fully leverages IE's feature set and can render sophisticated user interfaces. Scripter client populates the agent's display with information pertaining to the call, the customer, and the campaign, based on behavior defined in scripts.

Interaction Scripter API is a set of JavaScript extensions that support events, actions, and attributes. These elements manipulate objects in all environments, regardless of whether Interaction Dialer is also installed. *Predictive* actions, events, and attributes work only when Interaction Dialer is installed on the IC server. *Standard* actions, events, and attributes work with or without Interaction Dialer.

- **Events** are notification messages from the IC server that trigger script functions. For example, an event can provide notification that a queue on the server has changed. When a call is placed on a queue, this event changes the queue, generating an event message.
- **Actions** are messages from Interaction Scripter scripts to the server that trigger an action on the IC server.
- **Attributes** are data items passed by actions to the IC server. A Dialer attribute is data from a column in a database that is associated with a campaign.

For more information, see *Interaction Scripter* in the *Add-On Products* section of the IC Documentation Library.

Dialer IceLib API

Dialer IceLib API provides a set of classes for accessing Interaction Center outbound dialing functionality. Classes in the ININ.IceLib.Dialer namespace provide Workflow Monitoring, Dialer Agent Session Managementm and Dialer Call Handling. Classes in the ININ.IceLib.Dialer.Config Namespace access your Dialer server's configuration.

Web Service Extensibility using SOAP and XML

The widespread use of the Internet by business organizations has led to the development of *web services*—remote procedure calls that carry out some sort of data processing task, typically to return data of some sort.

Web services allow computers to request and receive information in much the same way that people use a web browser to open a web page. Web services employ the familiar *request/response* mechanism used by web browsers:

1. A *client* (e.g. web browser) connects to a *server* and passes a request (fetch a web page). The client then waits for the server to respond.
2. The server responds in one of two ways. It either returns the requested information, or it responds with an error message that tells the client why the request could not be completed.
3. Once the server has responded to the client, it closes the connection, discards all state information about the transaction, and listens for another request.

In the world of web services, the client is a computer program that asks a server (another computer program) to execute a method (*web service*) that returns data (instead of a web page). In order for the request/response model to work, messages must be formatted in a way that both computers can understand. Since there are millions of computers connected the Internet, which use many different operating systems, the format of remote procedure calls has been standardized using a technology called *SOAP*. SOAP stands for *Simple Object Access Protocol*.

SOAP is an XML-based wire protocol designed for decentralized, distributed networks such as the Internet. SOAP defines conventions that allow one computer to invoke a remote procedure in another. These remote procedure calls (SOAPrequests) can be transported using a variety of network protocols. HTTP protocol is widely used.

XML stands for *Extensible Markup Language*. XML provides a structured way to define data in plain text format, so that data can be exchanged between computers. SOAP messages are XML documents, which are just text files formatted according to some very specific guidelines. (SOAP is the specification that defines the guidelines used to describe remote procedure calls using XML.)

Before a SOAP request can be transported to another computer, the request is structured using XML so that the remote system can interpret the request in accordance with the SOAP specification. Responses from the remote procedure are returned as XML documents.

SOAP uses XML to package the data passed to a method, or received as a response. SOAP itself is nothing more than a set of rules that define how to describe method calls and return values using XML syntax. XML merely describes data, without consideration for the way that the data is processed or presented.

SOAP makes it possible for a remote computer to start a handler on the IC, and to receive data from IC in response.

Likewise, handlers that use the IC's SOAP tools can call remote web services.

SOAP extends IC interoperability to the entire Internet. Anything that "talks" SOAP through HTTP can communicate with the IC.

Any computer platform (Windows, Unix, Linux, Mac, etc.) that can create and transport a SOAP message request can start a handler on the IC. Depending upon the type of request, the handler may or may not send back a response containing values looked up by IC.

For example, a Unix Server might use *Enterprise JavaBeans* (EJB) to generate a SOAP Message requesting information about a user's status. When the request is received by IC, it starts a handler that looks up the user's status, generates a SOAP response, and transports the response back to the requesting server. When the Unix system receives this *SOAP payload*, it uses another EJB to parse and process the information.

Conversely, handlers created using the IC's *SOAP tools* can request data from web services and remote procedures. For example, a handler might request the current price of a stock from a brokerage service, check inventory levels from an inventory management system, conduct a credit card transaction, or obtain a weather report.

SOAP support on the IC is implemented by SOAP Tools in Interaction Designer (p. 67) that define initiators, invoke remote procedures, process requests and payloads. SOAP messages are channeled through a SOAP ISAPI Listener task that runs on an IIS server.

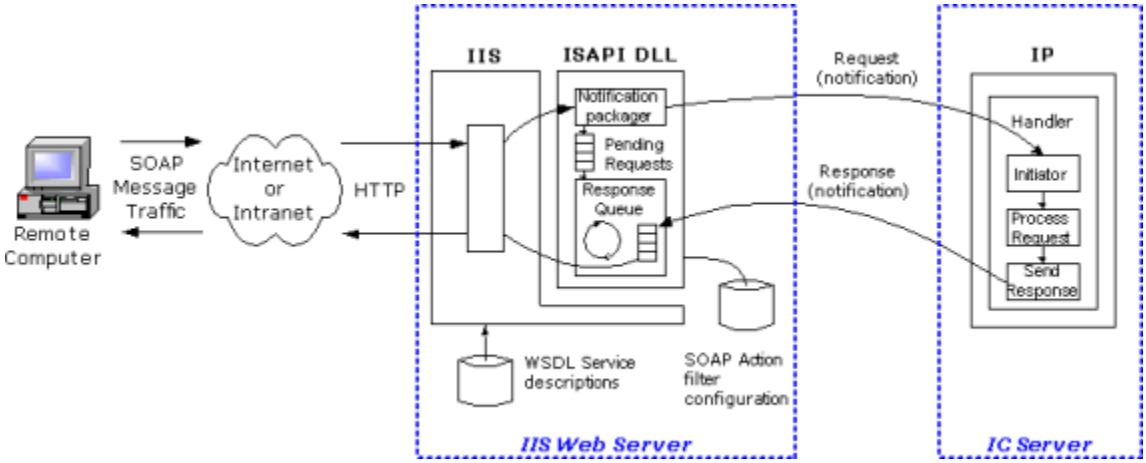
Developers can optionally use SOAP Notifier COM components (p. 70) to develop applications that directly invoke SOAP handlers. SOAP Notifier COM components are compatible with any language/application that supports Microsoft's Component Object Model.

HTTP protocol is typically used to transport SOAP message traffic (XML text files) across the Internet. In the IC environment, HTTP requests are received by *SOAP ISAPI Listener*—an ISAPI DLL that runs on an IIS web server. A *listener* receives incoming HTTP messages that contain SOAP requests for some type of service. It parses these messages, decides whether to process the request (based upon threshold values and filter configurations), and dispatches the request to the appropriate method for processing. If the service returns a response, the listener packages the response into an HTTP payload, and sends that back to the client. A listener also handles requests for WSDL information about web services.

- The *SOAP ISAPI Listener* looks at incoming SOAP requests, decides whether requests should be forwarded to the IC to invoke a handler, and forwards appropriate requests to IC's Notifier subsystem, which in turn calls Interaction Processor to invoke the handler associated with the initiator specified in the incoming message.
- The SOAP ISAPI listener and packages return values from handlers into outgoing HTTP responses, and sends them to the client. If the listener decides not to forward a request to the IC for processing, it returns a fault message (SOAP and/or HTTP) to the requesting client application.

SOAP Listener passes requests to the IC Notifier subsystem for processing. Notifier alerts the Interaction Processor subsystem, which in turn starts the handler needed to process the request. Response data from the handler is passed back to the Listener task for transport to the remote computer.

In general, SOAP Listener translates HTTP requests into notifications and acts as a gatekeeper to prevent denial of service attacks.



On the receiving end, the response message is decoded and used by the requesting computer in some way. This low-overhead approach permits a single server to share information with many clients. We've covered a lot of concepts in a short time, so let's review them:

- SOAP defines conventions needed to invoke the methods of a web service.
- A web service is a procedure of some sort that is called by a remote client application to initiate a process or request some sort of data.
- The IC's SOAP tools allow web services to be created using Interaction Designer.
- SOAP uses existing transport protocols (such as HTTP) to transmit an *XML payload* to another computer. The payload contains everything that the remote computer needs to execute a function (arguments and data).
- Services that understand SOAP requests can be expected to return XML responses in accordance with the rules of SOAP.

The Interaction Center supports open standards (SOAP, XML, WSDL, etc.) that promote interoperability and are applicable to many types of application development. SOAP tools are primarily used by developers, advanced handler authors, and professional services personnel. However, the *services* created using SOAP tools are another matter. Anyone, anywhere on the Internet is potentially a consumer or provider of information processed by SOAP handlers. The possibilities are limitless.

For example, an caller might enter a PIN number into an auto-attendant menu created using Interaction Attendant. In turn, Attendant could start a SOAP handler that passes the PIN number to a remote web service to look up information that is spoken back to the caller using the IC's text-to-speech capability.

A remote procedure invoked by SOAP can perform any kind of data processing tasks, ranging from a simple lookups to complex transactions that accept complex data types as input. SOAP does not impose any limits on the application functionality that can be invoked.

SOAP tools allow developers to create handlers that retrieve data from web services, or which function as web services. Handler-based services can be described using *Web Services Description Language (WSDL)*—an XML-based language that defines the functionality offered by a web service and how to access it. WSDL makes it possible to describe a service on the IC so that a worldwide audience can find and use it. WSDL describes the service, all parameters required to invoke it, and the location (endpoint) where the service can be accessed.

SOAP makes it possible for programs written in different languages and running on different platforms to communicate with each other. SOAP integrates the IC platform with business-to-business interactions and information services. Once a SOAP *endpoint* is exposed to the internet, a handler may call into the endpoint, which may be on the Internet or an Intranet.

SOAP is not appropriate for low-level, tightly-coupled transactions, due to network latency and the overhead imposed by the SOAP messaging encoding and decoding. SOAP is best suited for simple, high-level transactions, such as sending a name and PIN number to a service to obtain an account summary.

SOAP Tools in Interaction Designer

SOAP-related tools in Interaction Designer create handlers that process SOAP requests or responses. Each tool is briefly summarized.

- Initiator Tools (p. 67)
- Request Tools (p. 67)
- Payload Processing Tools (p. 68)
- Invocation Tools (p. 69)
- Helper Tools (p. 69)

Initiator Tools

SOAP Initiator

This initiator triggers if the "Notification Event" of the request matches a specified string. The Notification Event on which the Initiator triggers is specified in the property dialog.

Request Tools

SOAP Get Request Info

Queries some information from the request handle.

SOAP Abort Request

Aborts the request. Aborting a request is useful if a SOAP request handler is registered as a Monitor handler.

SOAP Get Transport Info

Returns an XML document containing transport specific (header) data. It allows the client to include any kind of out-of-band data in the request.

SOAP Expects Response

Takes a different exit path depending on whether the SOAP request requires a response (YES) or not (NO).

SOAP Parse Request Payload

Parses the payload of the request into an XML document.

SOAP Send Response

Sends the specified payload as response to the sender of the request. To support transport specific features, the "Transport Control Data" argument takes an XML node whose content will be sent back to the client. It can be used to send transport specific out-of-band data to the client.

Payload Processing Tools

SOAP Create Envelope

Creates a new SOAP envelope.

SOAP Get Body

Retrieves the Body element from the SOAP envelope. A body must exist. If it can't be found, the tool exits through "Failure" and attaches error information to the envelope.

SOAP Get Body Element

Retrieves the first body element that matches the given base name and namespace.

SOAP Add Body Element

Adds an entry to the body of the SOAP envelope.

SOAP Query Encoding Style

Matches a space separated list of URIs against the "encodingStyle" attribute of the given element. If the element doesn't have an 'encodingStyle' attribute, the parent of the element is checked and so on, until an element with an "encodingStyle" attribute is found. If that attribute contains any of the specified encoding style URIs, the tool returns through "Found" and returns the style that was found.

SOAP Get Header

Retrieves the header element from the SOAP envelope if it has one.

SOAP Get Header Element

Retrieves the first header element that matches the given base name and namespace. Returns the first element in the header if neither a name nor namespace is given. Takes "Not Found" exit if the envelope doesn't have a header or the element can't be found.

SOAP Get Header Elements

Returns iterator to a list of header elements filtered by the given arguments. Takes the "None" exit if envelope has no header or none of the header elements matched the filter criteria.

SOAP Add Header Element

Creates a header element and adds it to the given envelope. If the envelope does not have a header, one is inserted before the Body element.

SOAP Get Fault

Retrieves fault information from the SOAP envelope. If there is no <Fault> element in the envelope, the "No Fault" exit is taken and NULL elements and empty strings are returned.

SOAP Set Fault

Adds a <Fault> element to the envelope or replaces an existing one.

SOAP Create Fault Response

Copies the request envelope and replaces all children of the <Body> element with a single <Fault> element. It combines the functionality of the "SOAP Create Envelope" tool with the functionality of the "SOAP Set Fault" tool.

SOAP Get RPC Parameter

This is a convenience tool for cracking RPC requests. It retrieves a parameter element (child) from the first element in the <Body> element (method in an RPC request).

It returns the first element that matches all of the specified arguments.

SOAP Add RPC Parameter

This is a convenience tool for composing RPC requests or responses. It adds a parameter element to the first element in the body of the envelope, which represents the method in RPC requests. Use the XML tools to add complex data (not just a string) to the parameter by manipulating the returned "Parameter Element" node.

SOAP Get RPC Method Info

This is a convenience tool for cracking RPC requests. It retrieves the first child element of the SOAP <Body> element (Method element in RPC requests). It returns a collection containing the child elements of the method, which constitute the method arguments.

SOAP Get Next RPC Parameter

This tool returns the element node at the current iterator position and returns an iterator to the next position.

SOAP Create RPC Response

This is a convenience tool for composing the response envelope for an RPC request. It copies the source envelope and replaces the method element in the body with an element that has the same name but "Response" added to its name. It also adds a <Result> element as child of the method element.

SOAP Set Element Type

In SOAP, the type of an argument or the return value is specified by the service description and doesn't need to be included in the payload. However, the service may define the type as xsd:anyType, for VARIANT types. This tool allows you to include the type in the argument.

SOAP Create Array

Turns an element, for example an RPC parameter, into a SOAP array. The array is created for values supplied as list of strings or just a number of empty elements that can be populated with complex data.

Invocation Tools

SOAP HTTP Request

This tool issues an HTTP request to the specified URL with the SOAP request envelope as payload. The response body is parsed and returned as response envelope.

Helper Tools

SOAP Base64 Encode

Converts a supplied UNICODE string to the specified character set (default = UTF-8) and encodes the resulting data into a Base64 string. Characters that cannot be translated to the destination character set will be represented as '?'. Wide character sets, such as UTF-16 are currently not supported.

SOAP Base64 Decode

Decodes the base64 encoded string into the binary representation and converts it to UNICODE based on the specified character set.

SOAP Base64 Encode File

Reads the specified file as binary data and encodes it into a base64 string. This tool can be used to send any kind of data through SOAP requests. For example, you could encode a wave file in a SOAP message.

SOAP Base64 Decode To File

Decodes the base64 encoded string into the binary representation and writes the data to the specified file as binary data.

SOAP Notifier COM API

SOAP Notifier COM components provide a high-performance method of calling handlers without incurring the performance penalty of HTTP-based Listener operations. Third-party applications created using the SOAP Notifier COM components can directly create and forward packets to Interaction Processor, bypassing the need to create packets received using HTTP and the Soap Listener task. These packets are identical to those created by SOAP ISAPI Listener. However, the process is faster than HTTP-based Listener operations.

SOAP Notifier COM is appropriate for Windows client workstations that can run COM applications. It is not appropriate for operating systems (Linux, for example) that do not support COM. Interfaces in the SOAP Notifier COM API are listed below.

ISOAPNotifierTransport Interface

This component provides a COM wrapper on the Notifier (client) library. The Notifier Transport component represents a client connection to a certain Notifier server. It is used by the SOAP Request component as transport for requests.

Methods

Connect

Connects the transport to the Notifier server.

GetProperty

The GetProperty method retrieves a supplemental transport property.

SetProperty

SetProperty assigns a value to a supplemental transport property.

Properties

Connected (Get)

This read-only property returns True if the transport is connected to a Notifier server, meaning that the ISOAPNotifierTransport::Connect method has been called.

ISOAPRequest Interface

The ISOAPRequest interface issues raw Notifier SOAP requests. This component sends SOAP requests to a handler. It packages a request, sends it, waits for a response, and unpacks the response.

Methods

GetProperty

Retrieves a supplemental SOAP request property.

Initialize

Initializes the request object and sets the transport to be used. This method does not return a value.

Reset

Resets the request object by clearing SOAPAction, TransportInfo, and Payload data. This prepares the object for the next request.

Send

Sends the request and waits synchronously for the response object to be returned.

SetProperty

Sets a supplemental SOAP request property.

SetSOAPPayload

Sets the SOAP payload data to be sent with the request. Currently, this must pass an object that supports IStream. However, strings may be supported in a future release. This method does not return a value.

SetTransportInfo

Sets the transport information data to be sent with the request. Currently, this must pass an object that supports IStream. However, strings may be supported in a future release. This method does not return a value.

Properties

ExpectResponse (Get)

This property returns True if a response to this SOAP request is expected. False is returned if the request is a one-way request that does not generate a response.

ExpectResponse (Put)

Set this property True if the request should generate a response. If the request is a one-way request that does not generate a response, set this property to False.

InitiatorEvent (Get)

Returns the Initiator Event of this request. Uses SOAPAction if the event is not specified or empty. Changing the SOAPAction also resets this property.

InitiatorEvent (Put)

Sets the Initiator Event of this request. Uses SOAPAction if the event is not specified or empty. Changing the SOAPAction also resets this property.

SOAPAction (Get)

Gets the SOAPAction code of the request.

SOAPAction (Put)

Sets the SOAPAction code of this request.

Transport (Get)

The Transport property is read-only. It returns an IUnknown pointer to the transport object used for requests.

ISOAPResponse Interface

The ISOAPResponseInterface component writes payload or transport data into ISequentialStream streams, and returns payload objects.

Methods

WritePayload

This method writes payload data to an ISequentialStream passed as an argument.

WriteTransportCtrlData

This method writes transport control data to an ISequentialStream passed as an argument. If there is no transport control data, nothing is written, but the method succeeds.

Properties

Fault (Get)

Returns True if the server returned a SOAP Fault message; otherwise False.

Payload (Get)

Returns an IUnknown pointer to an object that implements IStream, so that you can access the object's SOAP response payload data.

RequestId (Get)

Returns the Identifier of this request.

Success (Get)

Returns True if the request was successfully executed on the server; otherwise False.

TransportCtrlData (Get)

This read-only property returns an object that implements IStream, so that you can access the object's transport control data.

ISOAPBase64 Interface

The ISOAPBase64 interface provides stateless methods that convert binary data to and from Base64 encoded strings. Base64 is an encoding system (defined by RFC 2045) that formats binary information for transmission through network connections.

Methods

DecodeToBinary

Decodes the base64 encoded data and returns it as a SAFEARRAY of bytes.

DecodeToFile

Decodes base64 encoded data and writes it to a file. Existing output files can be appended or overwritten.

DecodeToStream

Decodes Base64 encoded data and writes it into the supplied IStream or ISequentialStream.

DecodeToString

Decodes Base64 encoded data into a string.

Encode

Encodes the supplied data into a string that is Base64 encoded.

EncodeFile

Encodes the binary content of a file to create a Base64 encoded string.

SOAP-Related Documentation

The IC's SOAP support is primarily documented in a technical reference and installation guide titled *IC and SOAP API Developer's Guide*. This paper provides primers on SOAP and XML, and discusses components that must be installed to implement SOAP functionality in IC. Step-by-step procedures explain how to install and configure the SOAP ISAPI Listener DLL and SOAP Notifier COM components. The white paper also provides instructions for using the SOAP Tracer utility.

Conclusion

The Interaction Center provides a single, unified platform for third-party application development. Built-in tools such as Interaction Designer provide even casual developers with everything needed to develop powerful telephony solutions. These solutions can be exposed to the entire Internet as web services.

Robust APIs make it possible to create third-party applications that interact with Interaction Client, or replace it altogether. COM technology exposes the power of the Interaction Center platform and its subsystems (Interaction Designer, Interaction Client, Notifer, e-FAQ, etc.). Moreover, developers can extend server functionality to address any conceivable need, using C++ and other programming languages that support Microsoft's Component Object Model. Programmers can also create custom development tools that integrate seamlessly with Interaction Designer.

Since the IC is based on open standards, it supports the latest trends in software development, including COM, TAPI, and the ability to call web services on thousands of computers across the Internet. The system was designed from the onset to be powerful, open, and extensible. Third-party applications can integrate with IC using a variety of technologies, ranging from simple DDE actions, to complex COM component integration.

Glossary

Terms used by IceLib Developers

ACW

After Call Work.

ANI

Automatic Number Identification. ANI provides the phone number of a person calling in.

Attribute

An attribute is a piece of information about an object that travels with an object throughout IC. Telephone calls, chat sessions, and emails are all objects processed by the Interaction Center. An example of an attribute of an object might be the telephone number of the person who placed the call.

Some of the attributes that can travel with a telephone call are:

AssignedWorkgroup	Call Log	Call Note
Calling Party Number	Conference Chat	Customer Type
EIC_ACDWorkgroup	EIC_AdviceOfChargeEnd	EIC_anidnisstring
EIC_CallDirection	EIC_CallId	EIC_CallIDKey
EIC_CallingPNNumberDigits	EIC_CallingPNNumberingPlan	EIC_CallingPNScreeningInd
EIC_CallingPNTypeOfNumber	EIC_CallState	EIC_CallStateString
EIC_CallType	EIC_FormattedRemoteTn	EIC_LineName
EIC_LocalName	EIC_LocalTn	EIC_LocalTnRaw
EIC_LocalUserId	EIC_MonitoringUserId	EIC_PresentationIndicator
EIC_ReasonForCall	EIC_RecordingUserID	EIC_RemoteName
EIC_RemoteNameRaw	EIC_RemoteTn	EIC_RemoteTnDisplay
EIC_RemoteTNNormalized	EIC_RemoteTnRaw	EIC_StationName
EIC_UserSpecificQueueName	EIC_Workgroup	ForExtension
Hold	Language	LocalUserId
NoCall	OnHoldAudioFile	Recording RequestedBy

Some of the attributes of a chat objects are:

Address 1	Address 2	Address 3
Call Id	City	Company Name
Contact Description	Email	Info Request
Local ID	Name	Phone number
Queue name	Remote ID	State
StateString	Zip	

Since the IC is highly programmable, a variety of tools allow you to retrieve or modifiy the attributes of an object. CIC and MIC users can manipulate attributes using handlers. Interaction Attendant users can manipulate attributes using auto-attendant menus. And of course the Interaction Client COM API provides programmers with direct access to all object attributes.

It is the logical, event-driven architecture of the Interaction Center that makes this high level of integration possible. For a comprehensive guide to object attributes, see the Interaction Attributes Reference Guide (attrib.chm) that ships with the IC.

Blind Transfer

Transferring a call without speaking to the intended recipient is called a blind transfer.

Chat Session

A real-time typed conversation between two parties over the Internet.

Class

A template that defines the attributes of its members. The analogy of a "cookie cutter" is often used to describe what a class does. A cookie cutter is a template (class) that defines the shape and size of a cookie (object). An object is an instance of a class. For example, a cookie object is an instance of the cookie cutter class.

Callback Object

Callbacks are functions called when data changes. C++ provides hooks that allow functions to be called automatically when data changes.

Consult Transfer

Transferring a call after speaking with the intended recipient is called a consult transfer.

DND

Do Not Disturb

DNIS

Dialed Number Identification Service. DNIS is a feature of 800 and 900-number lines that provides the phone number of a person calling in.

DTMF

DTMF stands for Dual Tone Multi-Frequency. This impressive term describes the tones generated when buttons are pressed on a Touch-tone telephone. Each tone is actually a combination of two tones, one high frequency, and the other low frequency.

Interaction Center

The Interaction Center (IC) is a powerful Windows NT-based interaction-processing engine from Interactive Intelligence. This engine and its associated customization tools can be used to automate the processing of virtually any type of communications event including telephone calls, faxes, e-mail messages, pages and Web hits.

IUnknown Interface

Every COM component implements an internal interface named IUnknown. Client applications can use the IUnknown interface to retrieve pointers to the other interfaces supported by the component.

IDispatch Interface

The IDispatch interface provides a late-bound mechanism that can be used to access information the methods or properties of an object.

HRESULT (Result Handle)

COM interface methods return a value of the type HRESULT, which stands for "result handle". HRESULTs are 32-bit values with several fields encoded in the value. In Visual Basic, a zero result indicates success and a non-zero result indicates failure.

Object Watcher

An object watcher signals changes to a specific object. This is accomplished by the object calling back into the watcher. This process uses a separate thread, so you must handle GUI or thread dependent calls in the appropriate way for your development environment.

Park

In Interaction Client, calls can be "parked" on an intended recipient's extension until the recipient becomes available. Park places the call in the recipient's My Calls list so the recipient can select it later.

Status

The "My Status" combo box in the Interaction Client sets an agent availability indicator that affects the processing of calls directed to an agent. By default, any status condition other than Available, Available, Forward or Available, No ACD sends incoming calls to voice mail. Status values are maintained in Interaction Administrator.

Safe Array

A safe array is an array that contains information about the number of dimensions and the bounds of its dimensions.

WorkGroup

Workgroups are logical groups of users (e.g., departments) that can function as a group in the IC system. Workgroups can have extensions and queues that enable all members of a workgroup to receive calls notifying the workgroup. In addition, workgroups can receive regular calls and ACD calls that are routed to specific workgroups and agents. You may also create workgroups to serve as distribution lists (to the members) of voice mail, email, and faxes from within IC.

Terms used by e-FAQ Developers

Absolute Score Threshold

The absolute score threshold is a decimal number between 0 and 1. When searching for a match, e-FAQ ignores any entry whose absolute scores is lower.

Adjective

Adjectives are words that describe, identify, or quantify a noun.

Adverb

Adverbs are words that indicate how, when, where, how much, time, place, cause, degree, or manner.

Alias

The term alias represents a word and its expansion. For example "FAQ" expands to "frequently asked questions".

Each alias is a word, acronym, or phrase that you can map to words in e-FAQ's dictionary. e-FAQ's dictionary contains over one hundred thousand English words and synonyms. This dictionary handles most of the words that need to be matched. Occasionally, you may want to add your own terminology, or aliases, to this dictionary.

For example, if you map the FAQ alias to the phrase "frequently asked questions", when someone submits a query with the acronym FAQ, e-FAQ matches on the words frequently, asked, and questions.

Aliases

The term alias represents a word and its expansion. For example "FAQ" expands to "frequently asked questions".

Each alias is a word, acronym, or phrase that you can map to words in e-FAQ's dictionary. e-FAQ's dictionary contains over one hundred thousand English words and synonyms. This dictionary handles most of the words that need to be matched. Occasionally, you may want to add your own terminology, or aliases, to this dictionary.

For example, if you map the FAQ alias to the phrase "frequently asked questions", when someone submits a query with the acronym FAQ, e-FAQ matches on the words frequently, asked, and questions.

Alias Object

An alias represents a word and its expansion. For example "CIC" expands to "Customer Interaction Center". Aliases can either be applied when an entry is indexed, when a query is performed or both times. The point of when this expansion is applied can be configured for each alias separately.

Alias Objects

Each alias object represents a word and its expansion. For example "CIC" expands to "Customer Interaction Center". Aliases can either be applied when an entry is indexed, when a query is performed or both times. The point of when this expansion is applied can be configured for each alias separately.

Answer

An answer is the informational text in the body of an entry. When e-FAQ sends a response to a query, that response can contain entries, and thus, answers. Entries in a FAQ contain both questions and answers. When authoring a new entry, an answer is sent in the body of the email. You create answers when you create an entry.

Answer Score

When matcher compares a query to the entries in a FAQ, it assigns each question a score and each answer a score (depending upon how you have configured this behavior in e-FAQ Configuration Utility). e-FAQ then combines the two scores to calculate an entry score. The entries with the highest scores are sent in the response.

Attribute

An attribute is a value that denotes an entry as belonging to a subset of entries. You assign an attribute to an entry when you create that entry. For example, you can create an entry with attribute X. You can then configure a response mailbox to only search for answers among entries that have attribute X. When a query arrives in that response mailbox, only entries that have attribute X are considered as potential matches. Each entry may be assigned only one attribute.

Attributes

An attribute is a value that denotes an entry as belonging to a subset of entries. You assign an attribute to an entry when you create that entry. For example, you can create an entry with attribute X. You can then configure a response mailbox to only search for answers among entries that have attribute X. When a query arrives in that response mailbox, only entries that have attribute X are considered as potential matches. Each entry may be assigned only one attribute.

Author

The person who submits entries to a FAQ is the author of the entries submitted.

Authoring Mailbox

An email address to which authors submit entries. Each e-FAQ installation can have only one authoring mailbox.

Collection

A collection is a set of related objects. Collections contain an ordered set of items that can be referred to as a unit. Most collections allow you to access individual items by specifying either a numeric index or a string key. Collections are similar to linked lists, except that pointers are not used to refer items in memory. Collections can store any type of object or variable derived from a class.

Collections

A collection is a set of related objects. Collections contain an ordered set of items that can be referred to as a unit. Most collections allow you to access individual items by specifying either a numeric index or a string key. Collections are similar to linked lists, except that pointers are not used to refer items in memory. Collections can store any type of object or variable derived from a class.

Deictic Pronoun

Deictic means "pointing". Deictic pronouns always refer, or point to a context-specific subject. Deictic pronouns are a subclass of pronouns whose meanings are determined entirely by the context of a conversation. Deictic pronouns do not have a fixed meaning. In fact, all pronouns are somewhat deictic, since they all point to a noun.

For example, the pronoun "I" refers to the speaker, and "I" means different things depending on the speaker. "I" is usually considered to be less deictic than some other pronouns because it always refers to the speaker. Pronouns like "that" or "this" are very deictic since it is difficult to determine algorithmically what the pronoun points to.

For example, "that" sometimes refers to things discussed previously in the conversation. It can also refer to something being pointed to; or to a situation. For these reasons, it is difficult for computational linguists to process deictic pronouns programmatically.

Command

An instruction sent with an entry that tells e-FAQ to perform some operation with the entry. For example, the .DELETE command deletes the specified entry. The .FAQ command specifies the FAQ on which the operation should be performed. The .SPELLCHECK command tells e-FAQ whether or not to check the spelling of the entry before adding it to a FAQ. All commands are documented in the e-FAQ authoring help, which can be obtained by submitting an entry with HELP in the subject line to the authoring mailbox.

Dirty

When a feedback item, FAQ, entry or alias is dirty, it has been modified locally, and changes have not been sent to the server yet.

e-FAQ Configuration Utility

This application runs on the e-FAQ server. It is used by e-FAQ administrators to configure e-FAQ components:

FAQ Properties

Email Addresses for e-FAQ administrators, response mailboxes, email authoring, and others

Response Mailbox configuration

SMTP configuration

e-FAQ Knowledge Manager

e-FAQ Knowledge Manager is a web-based application that authors can use to create and manage entries and FAQs. Authors can also run reports to learn how their entries and FAQs are being used and maintained. See the e-FAQ 2.1 Knowledge Manager online help for more information.

e-FAQ Processor

The processor is the component of e-FAQ that watches for new queries and entries. If it sees a query has arrived, it either accepts or rejects the query based on several options configurable in the e-FAQ Configuration Utility.

If e-FAQ accepts the query, e-FAQ extracts a configurable number of words and passes the words to Matcher. Matcher returns zero, one, or more entries that matched the query. If no matches are found, Processor sends an email back to the user confirming that the message arrived, and then forwards the query to a human for further manual processing.

If Matcher returns one or more answers, Processor generates a response from one of the response templates. The response contains the entries that Matcher found. The response also contains a unique response identifier. If the user replies to the response, e-FAQ will see that the returned response contains a response identifier and forwards the query to a human for manual processing. Be sure to review the Flow Diagrams later in this document for a more detailed sequence of events. Several query-processing behaviors are configured in the e-FAQ Configuration Utility.

If Processor sees that an author has created a new entry, it extracts the question/answer pair. It then parses the question and answer, and attempts perform any other operations specified in the parameters set for that entry. Processor also performs delete operations and other actions on entries, sending a rejection message if it encounters errors.

Entry

An entry is a question/answer pair submitted by an author to a FAQ. An entry consists of question text and corresponding answer text and additional control information.

Entries can contain various attributes and parameters that e-FAQ uses to process the entry. These question/answer portions of the entries are sent in responses. See the e-FAQ™ 2.1 Knowledge Manager online help for more information on creating entries.

Entry ID

Each new entry is assigned a unique ID number when it is saved in the database. This Entry ID is often specified with other commands to identify which entry that an operation should be performed on. The Entry ID is included in the confirmation email sent to the author of the entry.

Entry Object

Data representing an entry in the FAQ. An entry consists of a Question text, its corresponding Answer text and additional control information.

Entry Objects

Entry objects represent an entry in the FAQ. An entry consists of a Question text, its corresponding Answer text and additional control information.

Expansion

An expansion is the expanded form of an abbreviation. For example, the word "USA" expands to "United States of America". Therefore the expansion of "USA" is "United States of America".

FAQ Database

An FAQ database is a collection of tables that hold entries and their attributes, such as question, answer, and entry ID. Matcher compares a query to the entries in the FAQ. You can create entries in the FAQ via the e-FAQ Knowledge Manager and the email-authoring interface. The FAQ database is

created during installation, and you can create new FAQs within the FAQ database using the e-FAQ Configuration Utility or the e-FAQ Knowledge Manager.

e-FAQ supports SQL Server 7 and MSDE (Microsoft Data Engine) for the FAQ database. MSDE is a special version of SQL Server designed to be embedded within an application such as e-FAQ. For large companies maintaining multiple FAQs, we recommend SQL Server 7 because it has a robust set of utilities for managing and backing up databases.

FAQ Object

Represents a FAQ, which consists of configuration data as well as a collection of Entries and Aliases.

Grade Threshold

When performing a query, e-FAQ assigns a grade to each entry. This grade is the number of standard deviations the absolute score of an entry is above the mean of the absolute scores of all entries. The grade threshold specifies the minimum grade accepted as result. This number must be in the range 1.0 to 10.0. The default value is 2.0.

IEnumVARIANT

The IEnumVARIANT interface provides a standard mechanism for enumerating a collection. In Visual Basic, the IEnumVARIANT interface is accessed using the "For Each" language construct. The VB example below demonstrates how to use IEnumVARIANT to iterate through all FAQs in a collection:

```
Private Sub Command1_Click()
Dim objServer As New EFAQCOMAPILib.Server
Dim objFaq As EFAQCOMAPILib.FAQ
' connect to the server
objServer.Connect MainForm.ServerName, "VbTestApp"

' utilize built-in IEnumVARIANT interface to display the name
' of each object in the collection
For Each objFaq In objServer.FAQs
     MsgBox objFaq.Name
Next
End Sub
```

The IEnumVARIANT interface is hidden and does not appear in Visual Basic's object browser.

Index

For each FAQ, e-FAQ contains an index of all the words contained in question/answer pairs that have been submitted with entries. e-FAQ parses the words contained in these entries, and calculates weights for those words.

You can configure whether to weigh words more heavily in the question or answer portion of an entry, or you can choose to weigh them both the same. You can choose to re-index a FAQ at any time using this API, the e-FAQ Configuration Utility or the e-FAQ Knowledge Manager. e-FAQ will tell pop a message when reindexing is required).

Indexing is necessary when you have changed Tokenize Numbers or Kill Words.

Note—Indexing puts a heavy load on your e-FAQ server and may degrade performance during the re-indexing process.

Indexing

For each FAQ, e-FAQ contains an index of all the words contained in question/answer pairs that have been submitted with entries. e-FAQ parses the words contained in these entries, and calculates weights for those words.

You can configure whether to weigh words more heavily in the question or answer portion of an entry, or you can choose to weigh them both the same. You can choose to re-index a FAQ at any time using this API, the e-FAQ Configuration Utility or the e-FAQ Knowledge Manager. e-FAQ will tell pop a message when re-indexing is required).

Indexing is necessary when you have changed Tokenize Numbers or Kill Words.

Re-Index

For each FAQ, e-FAQ contains an index of all the words contained in question/answer pairs that have been submitted with entries. e-FAQ parses the words contained in these entries, and calculates weights for those words.

You can configure whether to weigh words more heavily in the question or answer portion of an entry, or you can choose to weigh them both the same. You can choose to re-index a FAQ at any time using this API, the e-FAQ Configuration Utility or the e-FAQ Knowledge Manager. e-FAQ will tell pop a message when re-indexing is required).

Indexing is necessary when you have changed Tokenize Numbers or Kill Words.

KillWords

KillWords are words that are ignored when searching for an answer. Note—You must re-index the FAQ if you change this value once a FAQ contains entries. Be very judicial when specifying killwords. We recommend only to use a few of the very common "noise" words, such as a, an, the, it, is.

Killword

KillWords are words that are ignored when searching for an answer. Note—You must re-index the FAQ if you change this value once a FAQ contains entries. Be very judicial when specifying killwords. We recommend only to use a few of the very common "noise" words, such as a, an, the, it, is.

Matcher

Matcher is a Processor sub-component that uses artificial intelligence to match queries with answers (contained in the entries in a FAQ). Each entry in a FAQ contains a question and an answer. Matcher compares the query to each entry's question/answer pair and returns the ones that most closely match. Processor then takes a configurable number of highest-scoring entries and uses them in the response. You can configure some Matcher behaviors in the e-FAQ Configuration Utility.

Mean

The arithmetic average of a collection of numbers, computed by adding them up and dividing by their number.

Noun

Nouns are the names given to persons, places, things, or abstract ideas.

NULL

In C language, NULL denotes an empty object, or an object whose resources have been disposed of. In Java and Visual Basic, this is equivalent to object = nothing. For example:

```
If Not objAsyncError.NativeObject Is Nothing Then
    ' code statements go here
End If
```

Object

An object encapsulates data and functions that manipulate the data. Objects are accessed through a collection of methods (functions) that are exposed by an interface.

Query

A query is an email sent to a response mailbox, or a question passed to e-FAQ using the e-FAQ API. E-mail queries contain a question in the subject line or body.

Query Object

The Query object contains the parameters of a FAQ query. One can specify what FAQs are queried, the question to ask, the grade threshold and other parameters.

Query Result Object

A query result object represents one row in the result set.

Question

There are two types of questions in e-FAQ. The first is a question sent to a response mailbox. The second type of question is the question portion of an entry.

Response

One or more entries sent by e-FAQ in response to a Query. Responses are formatted with response templates.

Response Mailbox

An email address to which users send queries. e-FAQ parses queries sent to a response mailbox and tries to match that query to entries in the FAQ.

Response Template

e-FAQ transforms query result data with a response template to create a customized response for each user. In previous editions of e-FAQ, response templates were text files that contained substitution fields. Responses were generated using simple search and replace operations.

Server Round Trips

To transfer data to or from the e-FAQ server, DCOM remote procedure calls (RPCs) are used. RPCs are resource-intensive, especially if the server is not on the same machine as the client. Each exchange of information (round trip) between client and server entails marshaling the method arguments for transfer, transferring the data to the server, un-marshalling the data, invoking the method on the server, marshalling the data to be returned to the client, transferring the data, and un-marshalling the data.

1. When the server is local, the cost of the RPC call is mostly marshalling, un-marshalling and context switch.
2. When the server is remote, the time to transfer the data over the network is usually the biggest cost (time-wise).

You should coalesce as much data into one call as is possible, to reduce the number of RPCs required. This dramatically affects performance. Suppose, for example, that in the context of COM, the time it takes to do an in-process, intra-apartment as 1. A cross process call (server on same machine as client, but in different processes), takes approximately 3,000 times longer. A call to a remote machine takes at least 30,000 times longer than an in-process call.

Standard Deviation

Standard Deviation is the square root of the variance between numbers. In other words, for any set of numbers, the standard deviation is the square root of the difference between the mean (average) of the squares of the numbers and the square of the mean of the numbers.

Standard deviation evaluates the spread of data. For example, the numbers 3,7,1,17,2 are more spread out (scattered) than the numbers 5,6,6,7,6. Both sets of data have mean 6, but the first one has higher standard deviation. Low standard deviation indicates data generated by a consistent process. High standard deviation implies a greater difference between data elements.

Tokenize Numbers

If tokenization is enabled, e-FAQ considers numbers to be significant when calculating the answer score. For example, you would want to tokenize a FAQ that contains error messages, so that e-FAQ will consider the numbers important when searching for an answer. If a user asks, "What is error code 234?", e-FAQ will consider 234 to be a significant word.

If tokenization is disabled, e-FAQ ignores numbers. Suppose that a user submits "I've tried calling 2 times, and they suggested I email e-FAQ.", the digit "2" would be ignored when searching for an answer.

If you enable or disable tokenization after a FAQ contains entries, you must reindex the FAQ.

User

An e-FAQ user is someone who sends queries to a response mailbox.

UTC

UTC stands for coordinated universal time. UTC is also known as Greenwich Mean Time (GMT), Zulu time, universal time, and world time. UTC was developed for radio broadcasters whose transmissions can cross multiple time zones and international date lines. UTC time is maintained by a large number of atomic clocks around the world.

UTC uses a 24-hour system of time notation. "2:00 a.m." in UTC is expressed as 0200, and is pronounced, "zero two hundred." Fifteen minutes after 0200 is expressed as 0215; thirty-seven

minutes after 0200 is 0238 (pronounced "zero two thirty-eight"). The time one minute after 0259 is 0300. Each new day begins at midnight (0000) pronounced "zero hundred".

To convert UTC to local time, simply add or subtract hours from the UTC time. For persons west of the zero meridian to the international date line (which includes all of North America), hours are subtracted from UTC to convert to local time. The table below indicates the number of hours to subtract from local time zones (in North America) in order to convert UTC to local time:

To obtain Atlantic Daylight Time, subtract 3 hours from UTC
To obtain Atlantic Standard Time, subtract 4 hours from UTC
To obtain Eastern Daylight Time, subtract 4 hours from UTC
To obtain Eastern Standard Time, subtract 5 hours from UTC
To obtain Central Daylight Time, subtract 5 hours from UTC
To obtain Central Standard Time, subtract 6 hours from UTC
To obtain Mountain Daylight Time, subtract 6 hours from UTC
To obtain Mountain Standard Time, subtract 7 hours from UTC
To obtain Pacific Daylight Time, subtract 7 hours from UTC
To obtain Pacific Standard Time, subtract 8 hours from UTC
To obtain Alaska Daylight Time, subtract 8 hours from UTC
To obtain Alaska Standard Time, subtract 9 hours from UTC
To obtain Hawaii-Aleutian Daylight Time, subtract 9 hours from UTC
To obtain Hawaii-Aleutian Standard Time, subtract 10 hours from UTC
To obtain Samoa Standard Time, subtract 11 hours from UTC

UTC Date and Time Format

The format of a UTC Date/Time is: YYYYMMDDHHmmss.ss where:

YYYY= Year
MM = Month (00-12)
DD = Day (01-31)
HH = Hour (00-23)
mm = Minute (00-59)
ss = Second (00-59)
.ss = Decimal Second (00-59)

Example: 19990821133500.00

In this example, the date is August 21, 1999. The time is exactly 1335.

Verb

A verb declares something about the subject of the sentence to express actions, events, or states of being.

XML

XML stands for Extensible Markup Language. Most of the responses that e-FAQ sends to users are based on response templates. These templates are XML documents that contain:

- Substitution fields to hold values such as the response recipient's name, the matched entries, and other text.
- Conditional expressions for evaluating and formatting the entries in the response.
- Style definitions to format the responses.

XSL

XSL stands for Extensible Stylesheet Language. e-FAQ response templates may contain one or more embedded XSL style sheets that transform dynamically generated XML data of the query result or statistics into HTML to be inserted into the message body. The use of XSL to format dynamic data offers a very high degree of flexibility. For example, you can create templates that conditionally include or exclude entries based on user defined tags, or only include an abstract of the FAQ Entry and include a dynamically generated URL that links back to your website where the customer can retrieve the full answer.

Terms used with SOAP and XML Technology

COM

Microsoft's Component Object Model. The COM specification helps developers create component software that is compatible with a variety of languages, including C, ADA, Delphi, Java, and Visual Basic.

Denial of Service Attack

Denial of Service (DoS) attacks are attempts to overload a networked computer system so that it crashes, disconnects from the network, or becomes so overloaded that it cannot respond to legitimate requests.

DTD

Document Type Definition. A DTD defines the XML tags that can be used in an XML document, the order in which tags may appear, and limited information about data types. A DTD can be part of an XML document or can be referenced as an external file. The validating XML parser compares the DTD to the XML document and flags any errors. DTDs have been deprecated in favor of XML Schemas.

Handler

A program built in Interaction Designer that performs some action or actions in response to the occurrence of some event. A handler is a collection of steps organized and linked to form a logical flow of actions and decisions. Handlers are similar in structure to a detailed flowchart. Handlers can start other handlers called subroutines. A handler contains only one initiator step which identifies the type of event that will start the handler.

HTML

Hypertext Markup Language (HTML) is the markup language used to create World Wide Web pages.

IC Module

One of the many applications that make up the IC server. These applications have names like manager, server, and services. For example, Queue Manager, Fax Server, and Directory Services are all IC modules.

IDispatch Interface

The IDispatch interface provides a late-bound mechanism that can be used to access information about the methods or properties of an object.

Initiator

The first step in a handler that waits for a specific type of event to occur. When that event occurs, the Interaction Processor starts an instance of any handler whose initiator is configured for that event. An initiator is a required step that starts a handler. There can be only one Initiator in a handler. Initiator names describe the kind of event used to start a handler. Initiators can pass information from the event into variables that can be used within a handler. Subroutine initiators are not configured to watch for an event. Rather, they start when called from another handler.

Interaction Center (IC)

The Interaction Center Platform™ is a powerful platform for implementing comprehensive interaction management covering not only telephone calls and faxes but also e-mail messages, Internet text chats, Web callback requests, and voice over Net calls. Using the Interaction Center Platform, enterprises, contact centers, and service providers can centralize the processing of all customer interactions and provide a new level of service and consistency.

Interaction Designer

The IC graphical application development tool for creating, debugging, editing, and managing handlers and subroutines.

Interaction Processor (IP)

Interaction Processor is the event processing subsystem of the Interaction Center that starts instances of handlers when an event occurs.

IUnknown Interface

Every COM component implements an internal interface named IUnknown. Client applications can use the IUnknown interface to retrieve pointers to the other interfaces supported by the component.

Method

A method is a software subroutine that performs some type of data processing on an object in a computer system. Methods are sometimes called functions. Data can be passed when methods are called to perform some kind of work. For example, you might call a method named GetStockPrice and pass it a stock symbol to receive the current stock price as the return value.

Microsoft SOAP Toolkit

Microsoft's SOAP Toolkit makes it possible for programmers to invoke a web service as easily as invoking a method on an object. The Microsoft SOAP Toolkit reads in an WSDL file, and dynamically generates COM interfaces for operations described in the file. It packages method parameters in accordance with WSDL service descriptions.

Namespace

Since XML allows tags and attributes to be defined as needed, name collisions occur when the same name is assigned to a tag or an attribute, in different databases. For example, a teacher might define an element named "Grade" to represent a student's score. In the context of an agricultural operation, "Grade" could have a different meaning, as in "Grade A" eggs.

Namespaces resolve collision issues by associating XML attribute and element names with a specific context, or "namespace". A namespace is an identifier that helps computer programs determine whether identically named elements refer to the same type of data. Using namespaces, a program can determine that a data element named "Grade" in the "Schoolwork" namespace is different from an element called "Grade" in the "EggQuality" namespace.

Notifier

The IC module that acts as a communication center for all other modules. Notifier listens for events generated by other modules and notifies other interested modules that the event has occurred. Notifier uses a publish-and-subscribe paradigm.

Package

A SOAP package contains information needed to invoke a web service.

Payload

A payload contains data in XML format that is passed to or from a function. Request payloads contain everything needed to execute a function, including data and arguments passed as parameters. Response payloads contain the values that are returned from a function.

Processing Instruction

Processing instructions are read by application-level code (such as parsers) and are used to communicate information without changing the content of an XML document. For example, <?xml version="1.0"?> is a processing instruction that indicates that a document conforms to XML 1.0 specifications.

Processing instructions use <?target declaration ?> notation; where target is the name of the application that should process the instruction, and declaration is an instruction or identifier that is meaningful to the application. In the above example, xml is a reserved target that identifies XML parsers.

Protocol

A protocol is a set of rules that one computer uses to communicate with another.

Schema

XML Schema are the successor to DTDs for XML. XML schemas describe method calls, and can recognize and enforce data-types, inheritance, and presentation rules. A schema can be part of an XML document or can be referenced as an external file.

SOAP

Simple Object Access Protocol. SOAP is an XML-based protocol that requests or receives information from peer computers in a decentralized, distributed network. SOAP defines the minimal set of conventions that are needed to invoke code using XML and HTTP.

SOAP is used to invoke methods on servers, services, components and objects in another computer. SOAP specifies the XML vocabulary needed to specify method parameters, return values, and exceptions.

TCP/IP

Transmission Control Protocol/Internet Protocol.

Tool

The definition of a single action that can be performed within a handler. This definition includes name, label, runtime information (DLL and function), possible return codes, and parameters. Tools dragged into a handler become steps in that handler.

Valid

A valid XML document conforms to a document structure defined by a schema or DTD (Document Type Definition). Valid documents are well-formed documents that have a DTD or schema applied to them.

Vocabulary

A vocabulary is the set of tags and attributes that are used in an XML document.

Web Service

A web service is a method that can be invoked across the Internet. A web service can perform virtually any data processing activity, ranging from simple information lookups to complicated business transactions. SOAP is frequently employed to invoke web services.

Well-Formed

Well-formed documents follow the rules of XML.

WSDL

Web Services Description Language—an XML-based language that defines the functionality offered by a web service and how to access it. WSDL makes it possible to describe services on the IC so that a worldwide audience can find and use them. WSDL describes a service, the parameters required to invoke it, and the location of the endpoint where the service can be accessed.

XML

Extensible Markup Language. XML provides a structured way to define data in plain text format, so that data can be exchanged between computers.

XSL/XSLT

Extensible Style Language (XSL) is a specification used to transform XML documents into HTML. XSL Transformation (XSLT) provides similar functionality that transforms XML data into a different XML structure.

Revisions

This section describes what's new in IC 4.0.

IC 4.0 Service Updates 1-3.

There were no changes to this documentation, except for formatting and copyright updates.

IC 4.0

1. **Interaction Client Win32 COM API** (ClientCOM) has reached the end of its product life cycle. It is now formally deprecated. Developers must now use the **Interaction Center Extension Library** (IceLib) for applications development.

   IC 4.0 does not support the ClientCOM API. New client application features are implemented through IceLib. For information about IceLib, refer to the **IceLib API Developer's Guide**, in the System APIs section of the documentation library on your CIC server.

   The ClientCOM API is no longer available to install, nor is it supported by Interactive Intelligence. All references to ClientCOM were removed from this document.

2. Updated copyrights and trademarks in this document.

Copyright and Trademark Information

INTERACTIVE INTELLIGENCE (INTERACTIVE) HAS NO RESPONSIBILITY UNDER WARRANTY, INDEMNIFICATION OR OTHERWISE, FOR MODIFICATION OR CUSTOMIZATION OF ANY INTERACTIVE SOFTWARE BY INTERACTIVE, CUSTOMER OR ANY THIRD PARTY EVEN IF SUCH CUSTOMIZATION AND/OR MODIFICATION IS DONE USING INTERACTIVE TOOLS, TRAINING OR METHODS DOCUMENTED BY INTERACTIVE.

Interactive Intelligence, Inc.
7601 Interactive Way
Indianapolis, Indiana 46278
Telephone/Fax (317) 872-3000
www.ININ.com

Interaction Center Platform® Statement

This document may describe Interaction Center (IC) features that are not available or licensed in your IC product. Multiple products are based on the Interaction Center Platform, and some features are disabled or unavailable in some products.

Products based on the Interaction Center Platform include:

- Interactive Intelligence Customer Interaction Center® (CIC)
- Messaging Interaction Center™ (MIC™)

Since these products share some common features, this document is intended for use with all IC products, unless specifically stated otherwise on the title page or in the context of the document.

How do I know if I have a documented feature?

Here are some indications that the documented feature is not currently licensed or available in your version:

- The menu, menu item, or button that accesses the feature appears grayed-out.
- One or more options or fields in a dialog box appear grayed-out or do not appear at all.
- The feature is not selectable from a list of options.

If you have questions about feature availability, contact your vendor regarding the feature set and licenses available in your version of this product.