



Wave Client API User Guide

© 2011 by Vertical Communications, Inc. All rights reserved.

Vertical Communications and the Vertical Communications logo and combinations thereof and Vertical ViewPoint and Wave Contact Center are trademarks of Vertical Communications, Inc. All other brand and product names are used for identification only and are the property of their respective holders.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY

Vertical Communications, Inc. makes no representation or warranties with respect to the accuracy or completeness of the content of this publication and specifically disclaims any implied warranty of merchantability or fitness for any particular purpose, and shall not be liable for any loss of profit or any other commercial damage, including but not limited to, special, incidental, or consequential.

COPYRIGHT STATEMENT

This publication contains proprietary and confidential information of Vertical Communications, Inc. The contents of this document may not be disclosed, copied or translated by third parties, in any form, or by any means known, or not now known or conceived, without prior explicit written permission from Vertical Communications, Inc.

Vertical Communications, Inc. reserves the right to revise this publication and to make changes in content without notice.

Revision History

Release	Date	Documentation Changes	Page No.
2.5	10/11	GENERAL RELEASE	
		Added section "Adding logging to your custom application."	5-3
		Updated section "Programming tips and examples". Added: <ul style="list-style-type: none"> • "How do I log in and start a new session?" • "How do I sign a Contact Center agent in and out of a queue?" • "How do I store data with a party in a call?" • "How do I export a list of voice messages?" • "How do I locate a dialing service by its access code?" • "How do I find a specific Call object?" • "How do I check the availability or personal status of a user or extension?" • "How do I record conversations between Contact Center agents and external callers?" Updated: <ul style="list-style-type: none"> • "How do I create a call?" 	Starting on 5-4
2.0 SP1	04/11	SERVICE PACK RELEASE	
		The ViewPoint API is now called the Client API.	---
		Added section "License requirements" describing how licenses are used in custom applications.	3-2
		Added section "Obtaining and installing application certificates and license keys" including information on how to request an application certificate from Vertical.	4-4
		Added section ""Programming tips and examples."	5-4
2.0	09/10	GENERAL RELEASE	
		The separate <i>Vertical Wave Client API SDK Release Notes</i> document has been obsoleted. Client API-related information and issues are now covered in the <i>Wave IP Release Notes</i> and <i>Wave IP Known Issues</i> documents.	---
		Added section "Downloading and installing the Client API SDK".	3-2
1.5 SP3	09/09	SERVICE PACK RELEASE	
		Initial release of this document.	---

Contents

Contents

Chapter 1 Welcome to the Wave Client API

Who should read this guide	1-1
For more information	1-2

Chapter 2 The Client API object model

Typical applications	2-1
Client API object structures	2-2
Key to object group diagrams	2-3
The Session object	2-4
Structure of a session	2-4
The SystemConfiguration object	2-5
The UserItem object	2-6
The Folder object	2-7
How folders are used	2-7
Folder structure	2-9
Parent and child objects	2-10
Change tracking	2-10
Discarding changes	2-11
Accepting changes	2-11

	Object synchronization	2-11
Chapter 3	Getting Started	
	PC requirements	3-1
	License requirements	3-2
	Downloading and installing the Client API SDK	3-2
	Installing required and optional components	3-3
	Where to install the Client API SDK	3-4
	Installing the Client API SDK	3-4
Chapter 4	Using the Simple ViewPoint sample application	
	About the Simple ViewPoint sample application	4-1
	Getting help	4-2
	Installing the sample application	4-3
	Opening the sample application	4-3
	Obtaining and installing application certificates and license keys	4-4
	Obtaining application certificates and license keys	4-4
	Installing the application certificate on the Wave Server	4-5
	Installing license keys on the Wave Server	4-5
	Logging on from your application	4-6
	Using the sample installer program as a template	4-6
Chapter 5	Developing Custom Applications	
	Creating a new project	5-1
	Adding logging to your custom application	5-3
	Programming tips and examples	5-4

How do I log in and start a new session? ----- 5-5
How do I create a call? ----- 5-5
How do I find a specific Call object? ----- 5-6
How do I store data with a party in a call? ----- 5-7
How do I get events for any item? ----- 5-9
How do I create a Wave contact? ----- 5-9
How do I check the availability or personal status of a user or
extension? ----- 5-10
How do I change a user's personal status? ----- 5-11
How do I check the status of a Contact Center agent? ----- 5-11
How do I sign a Contact Center agent in and out of a queue? --- 5-13
How do I export a list of voice messages? ----- 5-14
How do I locate a dialing service by its access code? ----- 5-15
How do I record conversations between Contact Center agents and
external callers? ----- 5-15

Appendix A Client API Sample Application

SimpleViewPoint.Program ----- A-1

Appendix B Migrating Existing Applications From TeleVantage to Wave

New features in the Wave Client API ----- B-1
 Displaying ViewPoint objects' data in the user interface ----- B-1
 Session.Logoff() vs. Session.Dispose() ----- B-2
 Naming changes ----- B-2
 Distributing your Client API-based applications ----- B-2

Differences between the new and old Client APIs ----- B-2
 DCOM Security / Logging in a session ----- B-2
 Data change model ----- B-3
 Sinking events for items or collections ----- B-3
 Parent/Child relationship ----- B-4

Index

Welcome to the Wave Client API

The Wave Client API provides a rich interface that allows you to extend current ViewPoint functionality as well as integrate Wave with other enterprise applications.

The Client API contains the core objects upon which the ViewPoint application itself is built. Using the Client API, your custom application can do anything that ViewPoint can do. You can extend existing applications such as Outlook, Goldmine, Seibel, and Onyx, or write your own application from scratch.

To develop your application, you can use any development environment that uses standard .NET components, for example C#, Visual Basic .NET, and so forth.

The Client API is distributed as part of the Wave ViewPoint Software Development Kit (SDK), which also includes sample applications and documentation.

Who should read this guide

This guide is intended for anyone who is developing custom Wave applications using the Client API, including:

- Vertical Application Store application developers
- Vertical Wave Dealer/Development Partners who are developing custom applications
- End-user customers who are developing custom applications for their own use

For more information about the Vertical Application Store and the Dealer/Development Partner program, see the Wave Client API dashboard on V-Connect.

All custom application developers should read the following chapters:

- Chapter 2 provides an overview of typical Wave Client API applications, the fundamental objects and folders used, and some basics about how parent and child objects interact, how changes are tracked, and how objects are synchronized with information in the database.
- Chapter 3 lists the pre-requisites for the PC where the Client API SDK is installed, discusses custom application license requirements, and provides installation instructions.

- Chapter 4 discusses the Simple ViewPoint sample application included with the Client API SDK, including how to install the sample application and license included with the Client API.
- Chapter 5 guides you in creating your first project, and provides some programming tips and examples.
- Appendix A includes the sample application for reference.

If you are experienced with the TeleVantage Client API, read Appendix B, which describes features added in the Wave Client API SDK, as well as notable differences between the Wave Client API and the TeleVantage Client API.

For more information

See the Wave Client API dashboard on V-Connect for more about developing and licensing custom applications.

<http://vconnect.vertical.com/>

After logging in, choose **Products > Wave Client API**.

The Client API object model

CHAPTER CONTENTS

Typical applications	2-1
Client API object structures	2-2
The Session object	2-4
The SystemConfiguration object	2-5
The UserItem object	2-6
The Folder object	2-7
Parent and child objects	2-10
Change tracking	2-10
Object synchronization	2-11

Typical applications

Here are some examples of applications that you can create or functionality that you can embed in other applications using the Wave Client API:

- Your own ViewPoint GUI. Your application can do everything that ViewPoint does, or you can choose a subset of ViewPoint functions that fit your specific needs.
- A specialized operator's console designed to maximize an operator's ability to supervise the phone system.
- A GUI representing a Wave phone that looks and operates just like a real phone.
- An application that displays a list of people who are currently available to answer calls.
- A small application that changes your greeting every day, just for when friends call, or for all callers.

- A call tracking application that sends you notification of every call made, making it easy to track calls in a way that fits your specific needs.
- An application that synchronizes your Wave contacts with contact databases.
- A small utility that resides in the system tray and that allows you to change your Wave personal status.
- A small, lightweight call monitor that resides in the system tray.
- A voice mail export application. The API can collect your voice mail and export it in WAV format. Your application can then treat your messages just like any other WAV files, for example, archiving them, storing them in a database, or even converting them to MP3 format and downloading them to your MP3 player or PDA.

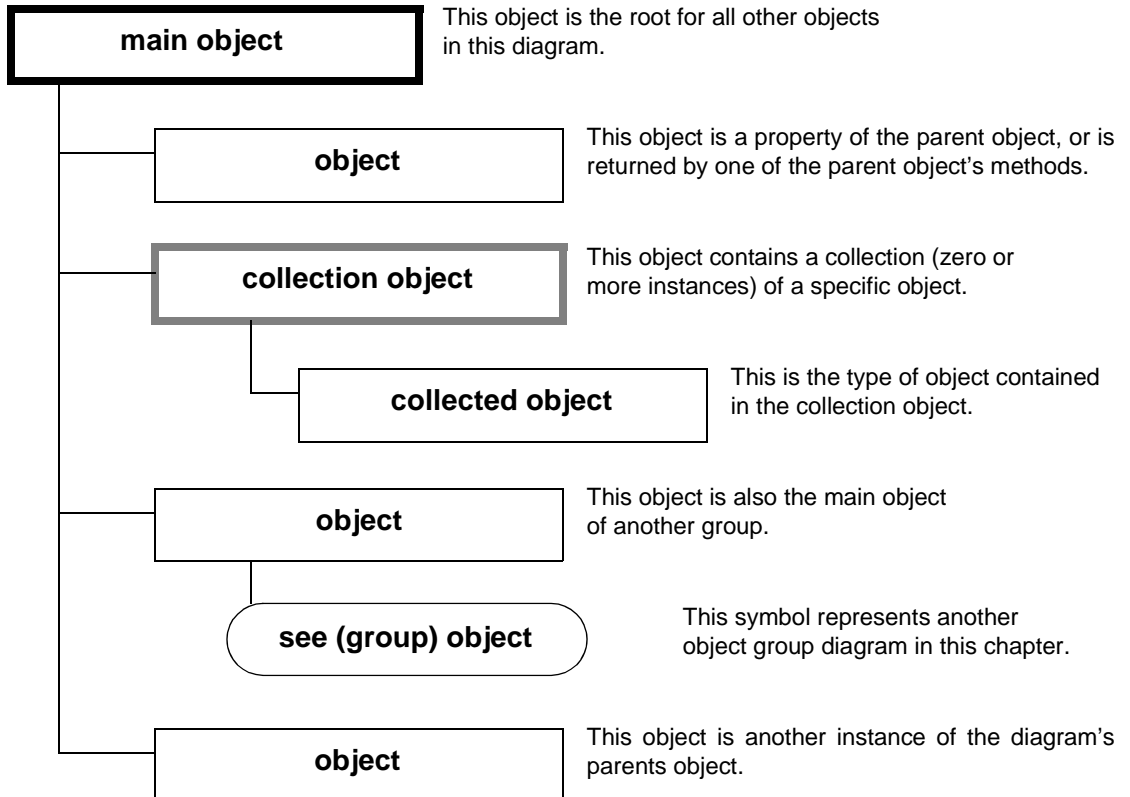
Client API object structures

Objects in the Client API are all members of about a dozen logical groups. This chapter summarizes the structure and function of some of those groups. The following groups are described in detail:

- The Session object. See page 2-4.
- The SystemConfiguration object. See page 2-5.
- The UserItem object. See page 2-6.
- The Folder object. See page 2-7.

Key to object group diagrams

Object group diagrams in this guide use the following graphic conventions:

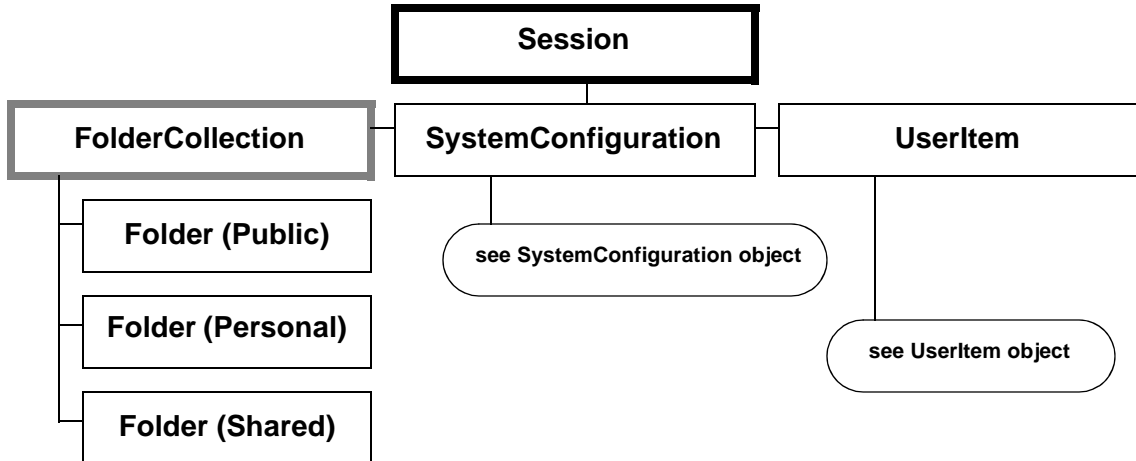


The Session object

Structure of a session

The Session object is the root of the Client API. It contains methods for logging on and off a Wave Server, and its properties include SystemConfiguration and UserItem objects. When a Session logs on to a Wave Server, the SystemConfiguration and UserItem objects are automatically populated with information about the current system and the current user's profile.

All other information is accessed from the Folders property on Session. By default the Folders object contains three Folder objects that cannot be deleted. These Folder objects have the following structure and are discussed in detail later (see “How folders are used” on page 2-7).

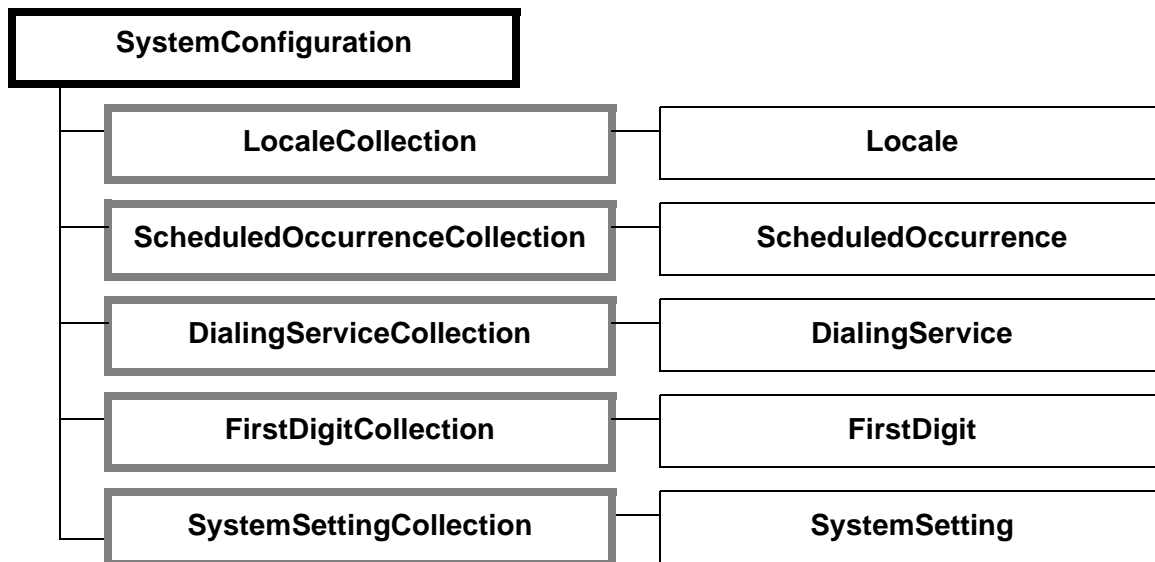


This structure uses the following objects:

- **Session object**
- **FolderCollection object**—A collection of Folder objects.
- **SystemConfiguration object**—Represents the system as a whole.
- **UserItem object**—Represents the currently logged in user.

The SystemConfiguration object

The SystemConfiguration object contains system-wide settings for a Wave ISM system. The SystemConfiguration object has the following structure:



The SystemConfiguration object structure has the following properties and objects:

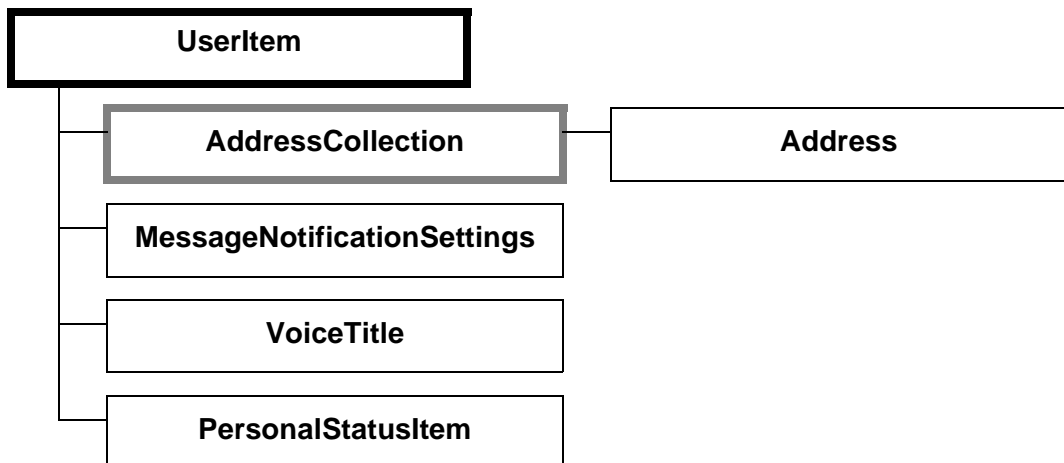
- **Locales property**—Returns a LocaleCollection object containing a list of Locale objects.
- **Locale object**—Contains a locale code identifying a Wave language locale setting.
- **ScheduledOccurrences property**—Returns a ScheduledOccurrenceCollection containing a list of ScheduledOccurrence objects.
- **ScheduledOccurrence object**—Contains a record of the times when pager and e-mail message notifications should be sent.
- **DialingServices property**—Return a DialingServiceCollection containing a list of DialingService objects.

- **DialingService object**—Contains information about an access code used to place an outbound call.
- **FirstDigits property**—Returns a FirstDigitCollection containing a list of FirstDigit objects.
- **FirstDigit object**—Contains information about a first digit used in the numbering plan such as the first digit type and the number of digits expected.
- **SystemSettings property**—Return a SystemSettingCollection containing a list of SystemSetting objects.
- **SystemSetting object**—Contains information about a generic named system setting.

The UserItem object

The Session User property represents the currently logged in user. Depending on Permission settings, a UserItem object may have permission to modify its own settings.

The UserItem object has the basic structure shown below. The object has many more properties and methods but this is a basic example of the properties that return some sort of object.



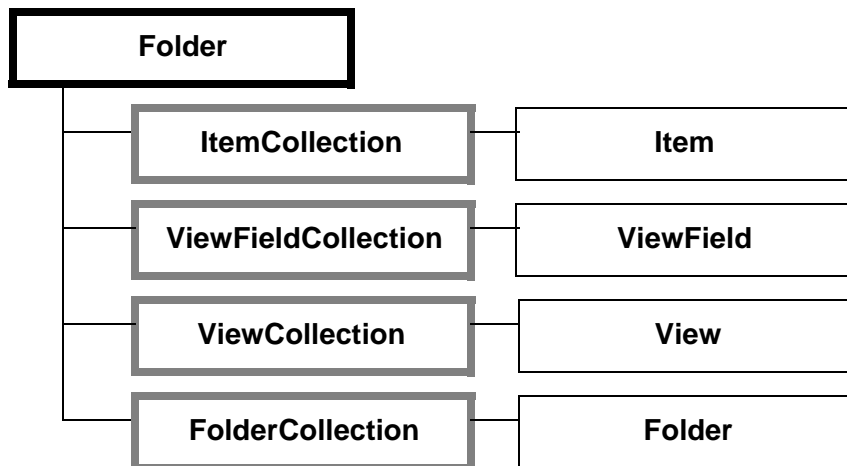
The UserItem object structure has the following properties and objects:

- **Addresses property**—Returns an AddressCollection containing a list of addresses.
- **Address object**—Contains information about addresses (phone number, extensions, and so forth) where a user or contact can be called.
- **PagerNotificationSettings, EmailNotificationSettings, CallNotificationSettings properties**—Returns a MessageNotificationSettings object that contains address and schedule information for sending pager, e-mail, or call message notifications to the user.
- **VoiceTitle property**—Returns an AudioClip object that contains information about the audio file representing the spoken name for the user. Also has the ability to play the audio over the speakers or phone.
- **LastAppliedPersonalStatus property**—Returns a PersonalStatusItem object that contains information on the last personal status that was applied to the user.

The Folder object

How folders are used

The Client API folder structure corresponds to the structure displayed in ViewPoint. Each folder used by ViewPoint contains a certain type of information and has a corresponding View that displays the information in rows and columns. The Client API encapsulates such Views in a Folder object structure:



A Folder object contains the following objects:

- **FolderCollection**—Each Folder object contains a FolderCollection object, which in turn can contain more Folder objects. Your application's Folder structure can contain as many levels as you wish.
- **ItemCollection**—Returns a collection of a specific type of object. For example, the Greetings folder in the example above would contain a set of GreetingItem objects. Each GreetingItem object would contain all of the information required for a specific greeting.
- **ViewCollection, ViewColumnCollection, and ViewFieldCollection**—Collections that provide you with information that you can use if you want to duplicate some of the predefined views used by ViewPoint. These collections are optional, and you do not need to use them in your application.

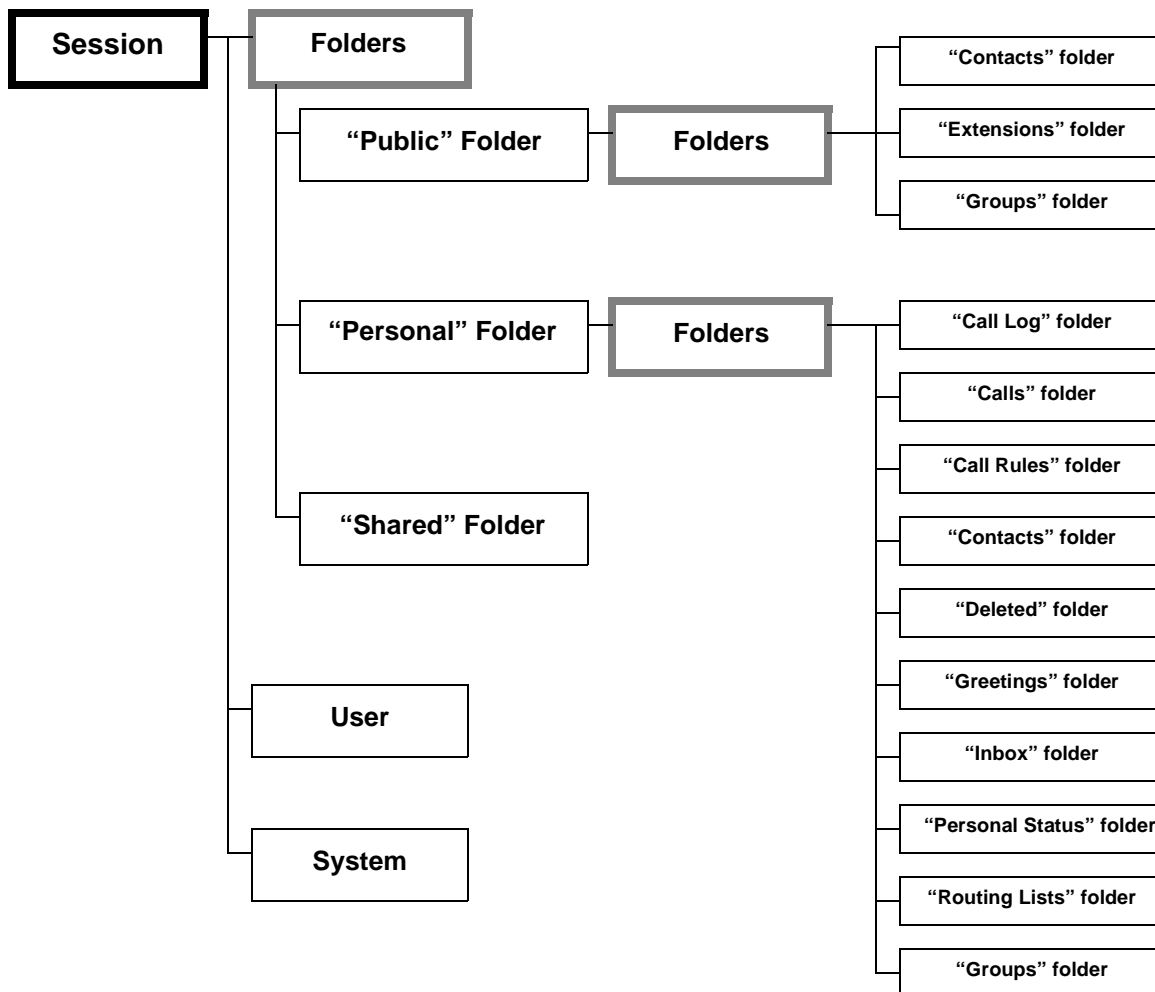
Folder types

To access a built-in folder, you call the Session GetBuiltInFolder method with the appropriate folder constant as the parameter. The following folders are available:

Folder name	Folder enumeration	Object collected (object structure diagram)
Call Log	CallLogs	CallLogItem objects
Call Monitor	Calls	CallItem objects
Call Rules	CallRules	CallRuleItem objects
Deleted	DeletedItems	MessageItem objects
Contacts	Contacts	ContactItem objects
Greetings	Greetings	GreetingItem objects
Messages	Messages	MessageItem objects
Personal Status	PersonalStatus	PersonalStatusItem objects
Routing List	RoutingLists	RoutingListItem objects
Saved	SavedMessages	MessageItem objects
Groups	Groups	GroupItem objects

Folder structure

The Folder objects diagrammed below are always present after a Session object logs on to the Wave Server. These objects are built in and cannot be deleted.



Parent and child objects

An object with properties that expose other objects is called a parent object and the objects exposed are called child objects. For example, the `UserItem` object exposes an `AudioClip` object via the `VoiceTitle` property - in this case `UserItem` is the parent object and the `AudioClip` is a child object. Child objects are loaded when their property is first accessed or when the parent object is about to be edited, either through a direct edit on the parent object itself or an edit on one of its loaded children (see “Change tracking” on page 2-10 for more information). Note that child objects are not loaded when the parent object is loaded.

When a parent object is disposed (by calling `Dispose`) all children of the parent object are disposed as well. This means that if you are holding a reference to a `UserItem` object and an `AudioClip` object for the user's voice title, and you then call `Dispose` on the `UserItem` object, the `AudioClip` object will also be disposed and its data will no longer be valid. If you then try to access properties on the `AudioClip` object after it has been disposed you will get an `ObjectDisposedException`.

Child objects cannot be saved directly - the only way to save a child object is to save its parent. When a parent object is saved all of its children are saved as well. If a child object has its own children they are also saved.

When parent objects are instantiated they are loaded with current data. This means that if you get a `UserItem` object, make changes, and then get another instance of the same `UserItem`, you will not see your changes because the data was loaded fresh. Child objects work differently - they get their data from their parent object. For example if you use the `VoiceTitle` property on `UserItem` to get an `AudioClip` object, make changes to that `AudioClip`, and then get the `VoiceTitle` property again, you will see the changes made in the other `AudioClip` object.

Change tracking

Objects in the Client API have a basic ability to keep track of changes and accept or discard those changes. The first time a change is made to an object (or `BeginEdit` is called on the object) the object will go into edit mode. When an object enters edit mode, all data for the object and its child objects are loaded.

There are two general types of objects for change tracking:

- **Item objects**—Based off `NotifiableObjectBase`. Called `SomethingItem`, for example `UserItem`.
- **Child objects**—Based off `ChildBase`.

Discarding changes

Both Item objects and Child objects support discarding changes. If at some point you decide that you do not want to retain the changes made to the object - but you do not want to load a new object reference - you can call `DiscardChanges`. All changes to the object and its children are undone and the object leaves edit mode without going back to the database for data.

Accepting changes

Child objects support accepting changes. If you want to retain the changes you've made to the object - but you do not want to save the object - you can call `AcceptChanges`. All changes to the object and its children are retained. If after calling `AcceptChanges` you then make more changes to the object, calling `DiscardChanges` returns the object and its children to the state they were in after you called `AcceptChanges`.

Item objects do not support accepting changes. If you want to retain the changes you've made to an item object, you can call `Save` on the object to save the updated data to the database.

Object synchronization

In the Client API, when an object is instantiated (for example, a `UserItem` object is instantiated when you access the `User` property on `Session`) it is loaded with data. If another Wave application then changes this data (for example, if the user is edited via the Global Administrator Management Console) the object will not automatically update with the new data in order to prevent any changes from being overwritten and to keep the object stable.

In order to tell if the data in the object is up-to-date, an object can subscribe for change notifications (by calling `Subscribe`), and then check a flag (the `Synchronized` property) to determine the state of the data.

By default an object will not be subscribed for updates and the `Synchronized` property will always return `OutOfSync` because the object does not know if it is current or not.

Once an object has subscribed for updates the `Synchronized` property can return the following values:

- **InSync**—The data in the object matches the current data in the database.
- **OutOfSync**—The data in the object does not match the current data in the database.
- **Deleted**—The object has been deleted (there is no longer any data in the database).
- **SyncRequired**—The object has been saved but not re-synchronized with the database.

For performance reasons objects are not automatically synchronized with the database when they are saved (by calling `Save`) because in many cases the application does not need the data back. If your application doesn't need the object after saving the new data (for example the application gets an object, brings up a dialog, saves changes on the dialog, and then closes the dialog) you can just call `Save`. If your application needs to continue using the object after saving it, calling `SaveAndSync` results in the object being updated with new data after the save.

If you call `Save` on an object and then try to call `Save` again without synchronizing the object, you get a `SynchronizeRequiredException` indicating that the object needs to be updated first.

Any time that an object is not synchronized, you can call the `Synchronize` method to load the latest data from the database into the object. Any changes you have made to the current instance of the object (and its child objects) will be lost when the current data is loaded.

Getting Started

CHAPTER CONTENTS

PC requirements	3-1
License requirements	3-2
Downloading and installing the Client API SDK	3-2

For the latest information about the Client API SDK, see the *Wave IP Release Notes*.

PC requirements

The following are required on the PC where the Client API SDK is installed:

- **Microsoft .NET Framework 3.5 Service Pack 1**—Download this service pack from Microsoft at the following location:
<http://www.microsoft.com/downloads/details.aspx?FamilyId=AB99342F-5D1A-413D-8319-81DA479AB0D7&displaylang=en>
- **Microsoft Visual Studio**—The Client API supports any .NET language (C#, VB.NET, and so forth), and any version of Visual Studio (2003, 2005, 2008, 2010). Visual Studio 2008 SP1 is recommended.

License requirements

Each custom application written using the Client API requires a license that identifies the application and application users to the WaveIP platform. Custom application licensing is handled by the Client API SDK's integrated licensing framework.

A license consists of the following:

- **Application certificate.** There is one application certificate per application that identifies the application to the Wave Server.
- **License key(s).** A license key allows a user to run the application.

Your license requirements depend on the type of application you are developing:

- A **Server-based application** allows unlimited users to run the application on the Wave Server.
- A **per-user application** allows a specific number users to run the application.

You use different licenses depending on where you are in the development process:

- **While getting started**—The license included with the Client API SDK is adequate for learning how to use the API and work with the sample application. This license is installed when you install the Simple ViewPoint sample application, as described on page 4-3.
- **During application development, testing, and deployment**—To fully develop and test your custom application (for example, to support multiple users), you need to contact Vertical to obtain the appropriate license for your application type. For more information, see “Obtaining and installing application certificates and license keys” on page 4-4.

Downloading and installing the Client API SDK

Important: Before you begin, go to V-Connect to check for critical information and to see if any additional required HotFixes have been released.

If you apply any new ViewPoint HotFixes on your Wave Server, you also have to upgrade the Client API SDK. See V-Connect for specific ViewPoint HotFix information, and to download the corresponding version of the Client API SDK.

Installing required and optional components

Perform the following steps on the PC where the Client API SDK will be installed:

1. **Install Microsoft .NET Framework 3.5 Service Pack 1.** Do one of the following:

- Download SP1 from the Microsoft Download Center at the following location:

```
http://www.microsoft.com/downloads/details.aspx?FamilyId=AB99342F-5D1A-413D-8319-81DA479AB0D7&displaylang=en
```

- On the Wave Server, run:

```
<Wave Server name>\NetSetup\ISSetupPrerequisites\Microsoft .NET Framework 3.5 SP1\dotnetfx35.exe:
```

2. **Install any version of Microsoft Visual Studio**—2008 (recommended), 2005, or 2003.

Important: Visual Studio 2008 SP1 is recommended because the Simple ViewPoint sample application included with the Client API SDK is written in C# with Visual Studio 2008. If you are using a prior version of Visual Studio, the Simple ViewPoint sample application project files will not open in Visual Studio but you will be able to create a new project and compile it.

The Client API supports any .NET language, including C#, VB.Net, and so forth.

3. **(Optional) Install the Windows Installer XML (WiX) toolkit.** The WiX toolkit is a free, open-source, professional quality installer toolkit. You can download WiX v3 from the following location:

```
http://sourceforge.net/project/showfiles.php?group_id=105970&package_id=168888
```

Note the following:

- Installing the WiX toolkit is optional, but you do not install it, you will not be able to use Visual Studio to open the sample installer program included with the Client API SDK. (Not installing the WiX toolkit will not affect your ability to compile the Simple ViewPoint sample application itself.)
- The Client API SDK sample installer program was written using the WiX toolkit and will install the compiled binaries and all dependencies for the Simple ViewPoint application on an end-user PC. You can also use the sample installer program as a template when creating an installer program for your own application.

For more about the Simple ViewPoint sample application, see page 4-1.

Where to install the Client API SDK

- You can install ViewPoint on the same PC with the Client API SDK and your own Client API applications.
- Do not install the Client API SDK on the Wave Server—this is not a supported configuration.

Installing the Client API SDK

To install the Client API SDK

1. Install the Client API SDK from the Wave Server. The Client API SDK installer is installed on the Wave Server in the FTP directory:

```
C:\Inetpub\ftproot\public\workstationapps
```

This is accessible from your ftp client (or browser) at:

```
ftp://<SERVERNAME>/public/workstationapps/
```

2. Run the Client API SDK installer program:

```
ViewPointApi.Sdk.msi
```

The Client API SDK will be installed in the following location:

```
C:\Program Files\Vertical Wave\ViewPoint SDKv2.0
```

Using the Simple ViewPoint sample application

CHAPTER CONTENTS

About the Simple ViewPoint sample application	4-1
Installing the sample application	4-3
Obtaining and installing application certificates and license keys	4-4
Using the sample installer program as a template	4-6

About the Simple ViewPoint sample application

The Simple ViewPoint sample application included with the Client API SDK was written in C# using Visual Studio 2008. If you are using an earlier version of Visual Studio, the SimpleViewPoint sample project files will not open in that version of Visual Studio, but you can create a new project and then compile it.

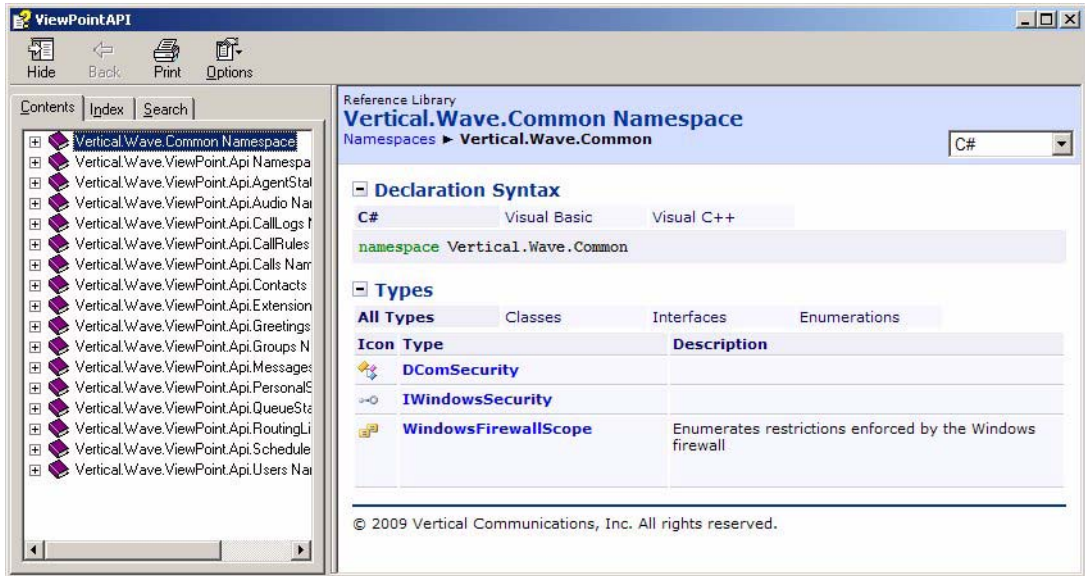
Tip: If you can build the SimpleViewPoint application successfully, that means that the Client API is installed correctly and is connecting to a Wave Server that is running the correct version of Wave ISM.

A copy of the complete SimpleViewPoint sample application is included for reference in Appendix A.

Getting help

Intellisense help is available while you are working on your application in Visual Studio.

You can also access the Client API Help by clicking **Start > Programs > Vertical Wave ViewPoint > ViewPoint Api Help**.



Installing the sample application

The Simple ViewPoint sample application installer program installs the sample application and adds the development application certificate to the Wave Server.

Important: The application certificate included with the Client API SDK is adequate for learning how to use the API and get started with basic application development, but to fully develop and test your custom application, you need to obtain appropriate the appropriate application certificate and keys from Vertical. For more information, see “Obtaining and installing application certificates and license keys” on page 4-4.

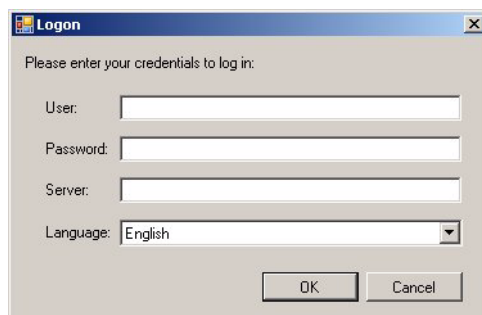
To install the sample application

1. Run `SimpleViewPointSetup.msi`. The default location is:
`C:\Program Files\Vertical Wave\ViewPoint SDK\v2.5\Samples\SimpleViewPoint`
2. When you are prompted to do so, enter the following license key:
`M9QC-XDTC-WW85-FA7L-6EB7-VIRB`

Opening the sample application

To open the sample application in Visual Studio

1. Click **Start > Programs > Vertical Wave ViewPoint > Simple ViewPoint Sample**. The Logon dialog opens:



2. Log on using valid ViewPoint logon credentials.
3. Click **OK**. The Simple ViewPoint application opens in Visual Studio.

Obtaining and installing application certificates and license keys

Each custom application written using the Client API requires a license that identifies the application and application users to the Wave IP platform. A license consists of an application certificate and associated license keys. See “License requirements” on page 3-2 for more about the Client API SDK’s integrated licensing framework.


Note: The steps in this section describe how to install application-specific application certificates and license keys. These application-specific licenses replace the license that is included with the Client API, and that is installed when you install the Simple ViewPoint sample application, as described on page 4-3.

Obtaining application certificates and license keys

To fully develop and test your custom application (for example, to support multiple users), you must obtain the appropriate application-specific licenses from Vertical.

- **If you are a Vertical Application Store application developer or a Vertical Wave Dealer/Development Partner** developing a custom application for the general market, you will be issued a unique application-specific license (either Server-based or per-user, depending on your application type).
- **If you are an end-user customer** developing a custom application for your own use, you will be issued a generic license (either Server-based or per-user). You can use the generic license for multiple applications.

To register your application with Vertical and request a unique Application Certificate

1. Go to V-Connect at.
<http://vconnect.vertical.com/>
2. After logging in, choose **Products > Wave Client API**.
 1. In the Technical documents section, click **Application Registration Template - For Certificate Request**.
 2. Click the  icon.
 3. Click **Download** and then save the file.
 4. Complete the form and send it to your Vertical Regional Sales Manager.

Installing the application certificate on the Wave Server

The application certificate is compiled into your application and identifies the application to Wave. You must install the application certificate on the Wave Server *before* any associated license keys can be installed.

There is a single application certificate for each application, and the same certificate must be installed on each Wave Server where the application will run. See the Main() function in Program.cs in the Simple ViewPoint sample application for an example of how to install the application certificate.

Note: In the current version, the only supported way to install a application certificate is via the Client API using the Session::InstallLicenseCertificate() method. This method only needs to be called once per Wave Server, but calling it multiple times will not cause any problems. In the Simple ViewPoint sample application included in the API, it is called by the application installer.

Installing license keys on the Wave Server

A license key allows a user to run an application. You can install license keys on the Wave Server *only after* the associated application certificate has been installed. Your application does not need to know anything about the license keys that authorize users to run the application. There is a one-to-many relationship between the application certificate and the license keys that are associated with it.

A license key can be installed in either of the following ways *after* the associated application certificate has been installed:

- **By calling the Session::InstallLicensekey() method** in the Client API. In the Simple ViewPoint sample application, this method is called by the application installer. This method only needs to be called once per Wave Server, but calling it multiple times will not cause any problems.
- **By entering the license key via the Software Licenses applet** in Global Administrator Management Console. See Chapter 24 in the *Wave Global Administrator Guide* for details.

Logging on from your application

The application passes the CERTIFICATE as a parameter of the Logon() method. The application does not need to know about the license key. If the certificate has a valid license on the Wave Server, then Login() will succeed. If there is no valid license, Login() will fail.

Using the sample installer program as a template

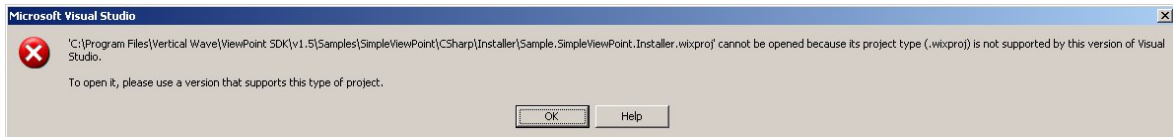
The Client API SDK includes a sample installer program that will install the compiled binaries and all dependencies for the Simple ViewPoint application on an end-user PC. You can use the sample installer program as a template when creating an installer program for your own application.

The sample installer program (`Sample.SimpleViewPoint Installer.wixproj`) was written using the Windows Installer XML (WiX) toolkit. See “Installing required and optional components” on page 3-3 for more about the WiX toolkit, include how to download it.

Note: Installing the WiX toolkit is optional, but you do not install it, you will not be able to use Visual Studio to open the sample installer program as described below. (Not installing the WiX toolkit will not affect your ability to compile the Simple ViewPoint sample application itself.)

To open the sample application in Visual Studio, click **Start > Programs > Vertical Wave ViewPoint > ViewPoint Api C# Samples**.

If the WiX toolkit is not installed on the PC where the Client API SDK is installed, the sample installer program will not load and you will see this error message:



Developing Custom Applications

CHAPTER CONTENTS

Creating a new project	5-1
Adding logging to your custom application.	5-3
Programming tips and examples	5-4

Creating a new project

Note: This example shows the steps to create a new project if you are using Microsoft Visual Studio 2008. There may be minor differences if you are using a different version of Visual Studio.

To create a new project

1. Open Visual Studio.
2. Choose **File > New > Project**.
3. Open a new C# Windows Forms Application.
4. Select the project in the Solution Explorer.
5. Choose **Project > Add Reference**.
6. Click the Browse tab and go to the bin directory for the Client API SDK. The default location is:

```
c:\Program Files\Vertical Wave\ViewPoint SDK\v2.5\bin
```
7. Select both of the following:
 - Vertical.Wave.ViewPoint.Api.dll
 - Vertical.Wave.ViewPoint.Api.Common.dll
8. Open `program.cs` in the code editor.

9. Add the following `using` line:

```
using Vertical.Wave.Common;
```

10. The `Main` function is decorated with the `STAThread` attribute. This attribute must be removed. To do so, delete or comment out the following line:

```
[STAThread]
```

11. Delete all the code from the `Main` function:

```
Application.EnableVisualStyles();  
Application.SetCompatibleTextRenderingDefault(false);  
Application.Run(new Form1());
```

12. Add the following code to the `Main` function:

```
static void Main(string[] args)  
{  
    // Initialize DCOM security for this application.  
    // This method must be called before initializing a session.  
    DComSecurity.Initialize();  
  
    try {  
Application.EnableVisualStyles();  
Application.SetCompatibleTextRenderingDefault(false);  
Application.Run(new Form1());  
    }  
    finally {  
// Terminate DCOM security for this application.  
// Call this method as the program exits.  
DComSecurity.Terminate();  
    }  
}
```

Adding logging to your custom application

By default, there is no ViewPoint API logging in a custom application. To enable logging, add or merge the following XML section into the configuration file for your custom application (ApplicationName.exe.config):

```
<configuration>
  <system.diagnostics>
    <trace autoflush="true">
      <listeners>
        <add name="Vertical.Logger"
            type="Vertical.Wave.Common.Logging.TraceListener,Vertical.Wave.Common, Version=2.0.1.XXXX, Culture=neutral, PublicKeyToken=b46a164ef129e6ed"
            initializeData="APPLICATION_NAME" />
        <remove name="Default" />
      </listeners>
    </trace>
  </system.diagnostics>
```

Where

- APPLICATION_NAME is a descriptive name for the log files.
- XXXX is the build number of Vertical.Wave.Common.dll located in the Global Assembly Cache.

The log files will be written to the following location:

```
C:\Users\<USERNAME>\AppData\Local\Vertical\ViewPoint\Logs
```

Where

- USERNAME is the name of the currently logged-in user.

Programming tips and examples

This section contains tips and examples that demonstrate how to perform some common application tasks using the Client API.

- How do I log in and start a new session? See page 5-5.
- How do I create a call? See page 5-5.
- How do I find a specific Call object? See page 5-6.
- How do I store data with a party in a call? See page 5-7.
- How do I get events for any item? See page 5-9.
- How do I create a Wave contact? See page 5-9.
- How do I check the availability or personal status of a user or extension? See page 5-10.
- How do I change a user's personal status? See page 5-11.
- How do I check the status of a Contact Center agent? See page 5-11.
- How do I sign a Contact Center agent in and out of a queue? See page 5-11.
- How do I export a list of voice messages? See page 5-14.
- How do I locate a dialing service by its access code? See page 5-15.
- How do I record conversations between Contact Center agents and external callers? See page 5-15.

How do I log in and start a new session?

```
// Create a new session object
Session _session = new Session();

// Validate the username/password and get the user's default station
ValidateLogonResult logonResult = _session.ValidateLogon(SERVERNAME,
USERNAME, PASSWORD, 0);
// Create a device to use with Logon()
StationInfo stationInfo = new StationInfo(logonResult.DefaultStation,
StationUsage.Owner);

// Log on to the session.
// "32402" is a unique number used to identify the application using
this Session object.
_session.Logon(SERVERNAME,           // Name of the server
              USERNAME,             // User name
              PASSWORD,             // Password
              ApplicationType.ViewPoint, // Application type
              32402,                 // Application id
              StationInfo;           // Device to use
              certificate);          // Certificate
```

How do I create a call?

To create a personal call to an external number

```
// Assuming _session was created and is already logged-in as shown
earlier

// create an address to call
Address address = _session.NewAddress("6175551234",
AddressType.PhoneNumber);

// call the address
_session.NewCall(address);
```

To create a personal call to a user

```
// Assuming _session was created and is already logged-in as shown
earlier

// Get the ExtensionItem object for the user to call.
ExtensionItem extensionItem;
_session.TryGetExtensionItem("109", out extensionItem);

// create an address to call
Address address = _session.NewAddress(extensionItem.Id,
AddressType.User);

// call the address
_session.NewCall(address);
```

To create a queue call

```
// Assuming _session was created and is already logged-in as shown
earlier

// create an address to call
Address address = _session.NewAddress("6175551234",
AddressType.PhoneNumber);

// call the address
_session.NewCall(address, _session.Agents[0].QueueId);
```

How do I find a specific Call object?

When dealing with inbound calls, you can use the Collection ItemAdded, ItemChanged, or ItemRemoved events to get the ID of each call. You can then pass the ID to Session.GetItem(), which will return the Call object. To determine which folder the call is in, use Call.Parent.

There can be multiple call folders. In ViewPoint, each call folder maps to the tabs at the bottom of the Call Monitor window. For example, Queue calls that haven't yet been routed to your logged-in user would be in the Queue's call folder as well as the personal folder. If you know which folder you want to work with, you can loop through the calls in that folder, as shown in the following example.

```
// Assuming _session was created and is already logged-in as shown
// earlier
// This sample code loops through the Calls folders so we can work
// with a specific call.
// Note that a single call might be in multiple folders (ex: Queue,
// Personal, and All)
FolderCollection allFolders = _session.AllFolders(ItemType.CallItem);
foreach (Folder callsFolder in allFolders)
{
    foreach (CallItem call in callsFolder.Items())
    {
        // <do something with the call>
    }
}
```

How do I store data with a party in a call?

CallItem::CustomData/CallParty::CustomData allow for custom data storage on a party in a call. For a conference call, CallItem::CustomData is a shortcut to store the custom data on the owner party, and in a 2-party call it is a shortcut to store the custom data on the “other” party in the call.

Once the custom data has been set, it can be retrieved using CallItem::CustomData/CallParty::CustomData. For a conference call, CallItem.CustomData is a shortcut to get the custom data from the owner party, and in a 2-party call it is a shortcut to get the custom data from the “other” party in the call.

Note: Data will not be immediately available after being set until the next Change event is received for the call.

To store custom data

```
// Assuming _session was created and is already logged-in as shown
// earlier

// Get the personal calls folder
Folder personalCallFolder =
_session.GetBuiltInFolder(BuiltInFolder.Calls);
```

```
// Subscribe for updates.
personalCallFolder.Items().Subscribe();
// Subscribe is asynchronous and call data will come in on
// another thread.
// For simplicity in this sample, just sleep to give the server
// time to send the call data
Thread.Sleep(3000);

// Assuming there is at least 1 call in this folder
CallItem callItem = (CallItem)personalCallFolder.Items()[0];

// Set and save some custom data
callItem.PrimaryParty.CustomData.Add("DataName", "DataValue");
callItem.Save();
```

To retrieve custom data

```
// Assuming _session was created and is already logged-in as shown
earlier

// Get the personal calls folder
Folder personalCallFolder =
_session.GetBuiltInFolder(BuiltInFolder.Calls);
// Subscribe for updates.
personalCallFolder.Items().Subscribe();
// Subscribe is asynchronous and call data will come in on another
// thread.
// For simplicity in this sample, just sleep to give the server
// time to send the call data
Thread.Sleep(3000);

// Assuming there is at least 1 call in this folder
CallItem callItem = (CallItem)personalCallFolder.Items()[0];

// get the custom data
string dataValue = callItem.PrimaryParty.CustomData["DataName"];
```


How do I get events for any item?

This example is specific for contacts, but you can do the same to get events for call history (new call log entries), voice messages, calls, or any other item in any type of folder.

```
// Assuming _session was created and is already logged-in as shown
earlier
// Get the default folder of the desired type of item (in this case
contacts)
Folder contactsFolder =
_session.GetBuiltInFolder(BuiltInFolder.Contacts);
// Get the collection of contact items from the folder
IItemViewCollection contactsItemViewCollection =
contactsFolder.ItemViews();
// Subscribe for change notifications
contactsItemViewCollection.Subscribe();
// Setup event handlers for each type of change you are interested in
contactsItemViewCollection.ItemAdded +=
contactsItemViewCollection_ItemAdded;
contactsItemViewCollection.ItemChanged +=
contactsItemViewCollection_ItemChanged;
contactsItemViewCollection.ItemRemoved +=
contactsItemViewCollection_ItemRemoved;
```

How do I create a Wave contact?

```
// Assuming _session was created and is already logged-in as shown
earlier
// Get the Contacts Folder.
Folder contactsFolder =
_session.GetBuiltInFolder(BuiltInFolder.Contacts);

// Create a new Contact object
ContactItem contact = (ContactItem)contactsFolder.NewItem();

// Set the Contact properties.
contact.FirstName = "Jim";
contact.LastName = "Williams";
contact.Pin = "123";
// Add a new Address object to the Contact.
Address address = contact.Addresses.NewItem();
// Set the Address properties for the new Contact.
address.AddressType = AddressType.PhoneNumber;
address.Category = AddressCategory.Home;
address.Description = "Jim's home number";
address.Value = "6175551234";
```

```
// Make sure the address uses the dialing rules defined in the Dialing
Service
address.AddressSubType = AddressSubType.UseRules;
address.IsDefault = true; // This will make this the default address
for this contact
// Call the Validate method so that all assigned fields can be checked
address.Validate(true);

// add the address to the address collection
contact.Addresses.Add(address);

// Save the new Contact.
contact.Save();
```

How do I check the availability or personal status of a user or extension?

To check the availability of the currently signed-in user

```
// Assuming _session was created and is already logged-in as shown
// earlier
string personalStatusDisplayName =
_session.User.LastAppliedPersonalStatus.DisplayName;
Availability availability = _session.UserAsExtensionItem.Availability;
```

To check the availability of other users

```
// Assuming _session was created and is already logged-in as shown
// earlier
// Get the user from the extension folder
Folder extensionFolder =
_session.GetBuiltInPublicFolder(BuiltInPublicFolder.Extensions);

// Filter the collection to only the (one) extension we care about
// (by name)
// (agent stats folder for queueId)
const string filterFormatString = "{{0}}={0:d}";
FilterStringBuilder filter = new FilterStringBuilder(
    String.Format(filterFormatString, "Jane User"),
    Fields.ExtensionItem.NameFirstLast);

ExtensionItemCollection extensionItems =
    (ExtensionItemCollection) extensionFolder.Items(null, filter);
ExtensionItem extension = extensionItems[0];
Availability availability = extension.Availability;
```

```
string personalStatusDisplayName = extension.PersonalStatusName;
```

How do I change a user's personal status?

The following example shows how to change a user's personal status via the API. One way to use this technique is to integrate this code with a program written using the Microsoft Outlook API that monitors scheduled events in Outlook. Whenever a scheduled meeting starts, the program automatically changes the user's personal status to "In a Meeting". When the meeting completes, the program then automatically changes the user's personal status back to "Available".

```
// Assuming _session was created and is already logged-in as shown
earlier

// Get the Personal Status Folder.
Folder personalStatusFolder =
    _session.GetBuiltInFolder(BuiltInFolder.PersonalStatus);

// Look at each Personal status object in the folder
// until we find "*In A Meeting".
// Note that the Name field of statuses are prefixed with "*"
foreach (PersonalStatusItem personalStatus in
personalStatusFolder.Items())
{
    if (personalStatus.Name == @"*In A Meeting")
    {
        // Use the ApplyPersonalStatus method with our
        // "*In A Meeting" Personal status object
        _session.User.ApplyPersonalStatus(personalStatus);
    }
}
```

How do I check the status of a Contact Center agent?

```
// Assuming _session was created and is already logged-in as shown
earlier

// Get the extensions folder
Folder extensionsFolder =
    _session.GetBuiltInPublicFolder(BuiltInPublicFolder.Extensions);

// Get the ExtensionItem object for the Queue.
ExtensionItem queueExtensionItem;
    _session.TryGetExtensionItem("401", out queueExtensionItem);
// Get the Queue Id
```

```
ObjectId queueId = queueExtensionItem.Id;

// Get the ExtensionItem object for the agent.
ExtensionItem agentExtensionItem;
_session.TryGetExtensionItem("101", out agentExtensionItem);
// Get the Agent Id
ObjectId agentId = agentExtensionItem.Id;

// Filter the folder collection to only the (one) folder we care about
// (agent stats folder for queueId)
const string filterFormatString = "{{0}}={0:d} AND ({{1}}={1:d} AND
{{2}}={2})";
FilterStringBuilder filter = new FilterStringBuilder(String.Format
(filterFormatString,
    ItemType.AgentStatsItem,
    queueId.UnderlyingType,
    queueId.DbId),
    Fields.Folder.ItemType,
    Fields.Folder.OwnedByUserType,
    Fields.Folder.OwnedByUserDbId);
FolderCollection allFolders = _session.AllFolders(filter);
// we found the Agent Stats folder
Folder agentStatsFolder = allFolders[0];

AgentQueueStatus agentStatus;

// Find the AgentStat Item for the agent by matching AgentID.
foreach (AgentStatsItem agentStat in agentStatsFolder.Items())
{
    if (agentStat.UserId.DbId == agentId.DbId)
    {
        agentStatus = agentStat.Status;
        break;
    }
}
```

How do I sign a Contact Center agent in and out of a queue?

```
// Assuming _session was created and is already logged-in as shown
earlier

// Get the ExtensionItem object for the Queue.
ExtensionItem queueExtensionItem;
_session.TryGetExtensionItem("198", out queueExtensionItem);
// Get the Queue Id
ObjectId queueId = queueExtensionItem.Id;

// Get the extensions folder
Folder extensionsFolder =
_session.GetBuiltInPublicFolder(BuiltInPublicFolder.Extensions);

// Get the Extension Item ID for the agent from the extension folder
// Filter the collection to only the (one) extension we care about (by
name)
const string extensionFilterFormatString = "{{0}}={0:d}";
FilterStringBuilder extensionFilter = new FilterStringBuilder(
    String.Format(extensionFilterFormatString, "Jane User"),
    Fields.ExtensionItem.NameFirstLast);
ExtensionItemCollection filteredExtensionItems =
    (ExtensionItemCollection)extensionsFolder.Items(null,
    extensionFilter);
ObjectId extensionItemIdForAgent = filteredExtensionItems[0].Id;

// Get the Agent Stats folder
// Filter the folder collection to only the (one) folder we care about
// (agent stats folder for queueId)

const string filterFormatString = "{{0}}={0:d} AND ({{1}}={1:d} AND
{{2}}={2})";
FilterStringBuilder filter = new FilterStringBuilder
    (String.Format(filterFormatString,
        ItemType.AgentStatsItem,
        queueId.UnderlyingType,
        queueId.DbId),
        Fields.Folder.ItemType,
        Fields.Folder.OwnedByUserType,
        Fields.Folder.OwnedByUserDbId);
FolderCollection allFolders = _session.AllFolders(filter);
// we found the Agent Stats folder
Folder agentStatsFolder = allFolders[0];
```

```
// Find the AgentStat Item for the agent by matching AgentID.
foreach (AgentStatsItem agentStat in agentStatsFolder.Items())
{
    if (agentStat.UserId.DbId == extensionItemIdForAgent.DbId)
    {
        if (agentStat.IsSignedIn)
        {
            agentStat.SignOut();
        }
        else
        {
            agentStat.SignIn();
        }
        break;
    }
}
```

How do I export a list of voice messages?

The following example exports a list of voice messages and then plays message over the phone via the API.

```
// Assuming _session was created and is already logged-in as shown
earlier

// Get the Messages Folder.
Folder messageFolder =
_session.GetBuiltInFolder(BuiltInFolder.Messages);

// Get the Messages View and export the column data as XML.
Vertical.Wave.ViewPoint.Api.View messageView = messageFolder.Views[0];
string exportString = messageView.Export(ExportType.Data,
ExportFormat.Xml);

// Write the exported data to an XML file.
string path =
System.IO.Path.GetDirectoryName(Application.ExecutablePath);
using (StreamWriter outfile = new StreamWriter(path +
@"\MessageData.xml"))
{
    outfile.Write(exportString);
    outfile.Close();
}
```

```
// Assuming we parsed the XML file for the object ID
// of a message (and stored it in messageID)
string messageID = "07010000001483";

// Get the actual message object from the system and play it over
// the handset.
MessageItem message = (MessageItem)_session.GetItem(new
ObjectId(messageID));
AudioClip audioClip = message.AudioClip;
audioClip.DeviceType = AudioDeviceType.Phone;
audioClip.Play();
```

How do I locate a dialing service by its access code?

```
// Assuming _session was created and is already logged-in as
// shown earlier
// loop through the services and match the AccessCode
foreach (DialingService dialingService in
_session.SystemConfiguration.DialingServices)
{
    if (dialingService.AccessCode == "9")
        break;
}
```

How do I record conversations between Contact Center agents and external callers?

Call.StartRecording, Call.StopRecording, Call.PauseRecording, and Call.ResumeRecording allow you to record both queue and non-queue calls. In the following example, targetExtensionItem specifies the user's Inbox to which the recording will be sent. The time stamp and name of the person who did the recording are included as well. All recordings are sent to the specified user's Inbox—you cannot send recordings to a different folder. If you pass a timeout parameter of -1, the recording will not stop until some outside event occurs such as a StopRecording. terminationDigits allows you to specify a key (for example, #) that stops the recording when the key is pressed on the keypad.

```
callItem.StartRecording(targetExtensionItem,
CallRecordingFormat.PCM8K, -1, String.Empty, false);
```

Important: This example uses the default call recording format `CallRecordingFormat.PCM8K`. If you use a different call recording format, you will not be able to play back the recording from ViewPoint or convert the recording to a .WAV file in order to forward it via e-mail. You should use one of the non-default call recording formats only if your application itself manages playback.

Client API Sample Application

This appendix contains a reference copy of the sample SimpleViewPoint application included with the Client API SDK. The sample application is installed in the following default location:

```
C:\Program Files\Vertical Wave\ViewPoint SDK\v2.0\  
Samples\SimpleViewPoint
```

SimpleViewPoint.Program

```
using System;  
using System.Windows.Forms;  
  
using Vertical.Wave.Common;  
using Vertical.Wave.ViewPoint.Api;  
  
namespace Vertical.Wave.ViewPoint.Samples.SimpleViewPoint  
{  
    static class Program  
    {  
        //  
        // The Main method of a Wave application must not be a  
        // STAThread because // DCOM security must be initialized and  
        // ended manually, with the  
        // DComSecurity.Initialize() and DComSecurity.Terminate()  
        // methods.  
        //  
        /// <summary>  
        /// Main entry point  
        /// </summary>  
        /// <param name="args">CLI args</param>  
        // [STAThread] // <-- Must not use this attribute  
        static void Main(string[] args)  
        {  
            //  
            // Initialize DCOM security for this application.  
            // This method must be called before initializing a session.
```

```
//
DComSecurity.Initialize();

try
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    // Gather logon information from the logon dialog
    LogonInformation logonInfo = LogonManager.GetLogonInformation();

    // If the dialog was canceled then exit the application
    if (logonInfo == null)
        return;

    // Create a new session object
    Session session = new Session();

    //
    // Attempt to create a Windows Firewall exception for this
    // application.
    //
    // This is only required if you want to avoid the default
    // Windows Firewall exception dialog. If so, it must be called
    // after you create a session but before logging on.
    //
    switch (WindowsFirewall.AddException(session, logonInfo.Server,
        Application.ProductName, true))
    {
        case WindowsFirewallExceptionResult.AddedRebootRequired:
            {
                // Should reboot here
                break;
            }

        case
        WindowsFirewallExceptionResult.FailedMustUpgradeToNewerWindows
        Firewall:
            {
                // Should inform the user that they should upgrade
                // to Windows XP SP2 or greater
                break;
            }
    }
}
```

```
// The key that goes with this certificate is:
// M9QC-XDTC-WW85-FA7L-6EB7-VIRB
const string certificate =
    "eSL2lw/IsDXNdakBvnCGNSmBEnQa1ch356K+JS4eR9jiTcDfR7lSVALXFdydn
tXsfMSK8F9f6hRKIBZ8J8lZFh/xrhfSXT3dav419olwkbZufgITkBzSLkaxRib
ajCUdAqUwYBb1LJASyZeokM67Fk4VMYfuNv/zxLDBBIuaFhzw8YefU17hBKJer
kES4e/wdlp0AQcQIR8ubBeghmwa9iYPz6MVLlfjNPRQNKfYDxRrww5OCK0y4sj
qMBxxwG09MwlpF0Xc/3yDRVMLuuU2TbFxxqs47pkLYndwGpVSmvDFczQF18lzf
fCdppCvB1QjsKYPugvLHfLy51GC4WdRjrYHMKsWa5kviKmg96biPkO+rKT1L1V
OSKj3cJmdtx796JAnKRB+a+SpZ0SIyEb/uEsjMu9jLyovFkAKtRNSQ2V7mih73
/7b64ndPkoxdUW5yg9Epg04Ef4AQfbAfpuJYuDqDagbN0ej3U6PP8FZK1zD8Tz
LhWqtPMLyim6Rh95Lg94YOBu+O81Vy6OwL/kknxL9AVseISXRzCs7tsnYeYho0
PcaBmnCfGizz9dyRTIgxxs6tOCXj35xWpCVBBKcsaDYaMNJ7uOogrKJZwZeXOU
aSEdedqqKziKtch6zxqPG";

// Log on to the session.
//
session.Logon(logonInfo.Server,           // Name of the server
              logonInfo.UserName,        // User name
              logonInfo.Password,        // Password
              ApplicationType.ViewPoint, // Application type
              0),                        // Application ID
              certificate);              // Application type

//
// Perform operations on the session object here. For this
// sample a new form will be displayed using the Session object
// that was just created and authenticated.
//
Application.Run(new MainForm(session));

// Make sure to log off from the session before exiting the
// application, otherwise certain processes will remain running.
session.Logoff();
}
finally
{
    //
    // Terminate DCOM security for this application.
    // Call this method as the program exits.
    //
    DComSecurity.Terminate();
}
}
```


Migrating Existing Applications From TeleVantage to Wave

This appendix discusses the following topics to help you migrate existing applications written using the TeleVantage Client API to run under Wave.

- New features in the Wave Client API
- Notable differences between the Wave Client API and the old TeleVantage Client API

New features in the Wave Client API

Displaying ViewPoint objects' data in the user interface

The Client API now provides the ability to utilize the .NET binding functionality to easily display objects' data in the user interface:

- **ItemView**—A light-weight, read-only snapshot of data for its corresponding Item. Unlike an Item, ItemViews cannot be subscribed to or synchronized.
- **ItemViewCollection**—Represents a collection of ItemView objects. It implements `IBindingList`, so it can be used as a binding source for the user interface.

For an example and further information, see `Sample.MainForm.LoadGridView`.

Additionally, all Client API Item objects implement `IPropertyChanged` and `IEditableObject`. This provides the ability to bind to them as well as have the ability to commit or rollback changes made to them.

Session.Logoff() vs. Session.Dispose()

When you want to log your Session object off from the Wave Server, you can now call Session.Logoff() or Session.Dispose(). As a general rule, use the Session.LogOff() method when you want to reuse the Session object and Session.Dispose() when you are done with it.

- **Session.Logoff()**—Disconnects the session from the Wave Server and disposes of all its underlying objects and state.
- **Session.Dispose()**—Calls Session.Logoff() and additionally allows the Session object to be cleaned up by the .NET Garbage Collector. This means that if you want to log into the Wave Server again, you need to create a new Session object first.

Naming changes

Some default Folder names have changed:

- The System Targets folder has been renamed to Extensions.
- The Workgroups folder has been renamed to Groups.
- The Call History folder has been renamed to Call Log.

Distributing your Client API-based applications

The Client API SDK now includes the Windows installer merge modules which should be used to distribute your API-based application to runtime systems. These merge modules contain the entire Client API runtime as well as any runtime files the API itself depends on.

The SDK also includes a sample installer program which uses these merge modules to install the SimpleViewPoint sample application on a runtime system. See “About the Simple ViewPoint sample application” on page 4-1 for more information.

Differences between the new and old Client APIs

DCOM Security / Logging in a session

Old Way—DCOM security was set for API-based applications by applying registry files. These settings would have to be applied as part of the application's distribution.

New Way—Application of registry files is no longer needed as long as you incorporate the calls to DComSecurity.Initialize() and DComSecurity.Terminate() in the Main() entry and exit points respectively in the distributed assembly.

For an example, see Main() in Program.cs of the Simple ViewPoint sample application.

Data change model

Old Way—ViewPoint items' data was always up-to-date and synchronized with the database. The objects' data could never get “dirty”. A side effect of this was that the data could be changed out from underneath you while you were manipulating an object.

New Way—There is no automatic way to enable the old behavior. Instead, ViewPoint items need to be explicitly subscribed to using the `Subscribe()` method in order for an application to get notified of adds, updates, moves, and deletes. When an application is notified of a change (using the provided events), the application can then request that the item be synchronized, using the `Synchronize()` method. Once `Synchronize` completes, the data is synchronized with the database. If you do not want an object to be subscribed, you can also manually synchronize the object to obtain the latest data.

ItemCollections on the other hand are similar in that they need to be explicitly subscribed to, again using `Subscribe()`, but each time you get an object from the collection it will be synchronized with the database. Again, if you do not wish for the collection to be subscribed, you can also manually synchronize the collection to get the latest data for the entire collection.

Sinking events for items or collections

Old Way—If you implemented the Event Handler then as soon as you accessed an ItemCollection, you would automatically get all events for items in that collection. Once you accessed the collection, you would always get events during the lifetime of the logged-in session, with no way to turn it off.

New Way—You can now use the .NET event subscription model to turn events on and off for objects and collections. You have to call `Subscribe()` on an object or collection in order for events to be fired, but you are also able to turn these events off via the `Unsubscribe()` method.

This new subscription model gives API developers a higher level of control over the amount of data that gets sent over the wire between the client and server. If you do not care about whether or not an item's data has changed, then you do not need to subscribe to those change notifications. If you need to know about changes to a particular Item or Collection only during a certain time period, you have the ability to selectively “listen” for those changes during those time periods. Since subscribing to and handling these events requires more processing overhead, the ability to unsubscribe provides a performance benefit.

Important: If two code paths are talking to the same instance of a collection object and have independently subscribed to the .NET events on the collection, the `Subscribe()` and `Unsubscribe()` methods are shared—not tracked independently for each code path. This means that if either code path calls `Subscribe()` or `Unsubscribe()`, all the events—in both code paths—will begin firing or cease firing respectively.

See page 2-11 for more about object synchronization. For an example, see `User_Changed()` in `MainForm.cs` of the Simple ViewPoint sample application.

Parent/Child relationship

Old Way—The old version of the Client API did not have a way of discarding changes on child objects without also having to discard their parents.

New Way—The Client API now provides methods on child objects that allow more granular control over managing changes to them independent of their parent.

See the following ChildBase method descriptions in the Client API online Help for details:

- `BeginEdit()`
- `CancelEdit()`
- `AcceptChanges()`
- `DiscardChanges()`
- `ResetChanges()`

See page 2-10 for more about parent and child objects.

Index

A

about

- Client API, 1-1
- object group diagrams, 2-3
- sample installer program, 4-6
- Simple ViewPoint sample application, 4-1

application certificates and license keys

- obtaining, 4-4

C

change tracking, 2-10

Client API

- about, 1-1
- application certificates and license keys, 4-4
- change tracking, 2-10
- downloading and installing, 3-2
- folders
 - folder structure, 2-9
 - how folders are used, 2-7
- getting help, 4-1
- license requirements, 3-2
- objects
 - Folder, 2-7
 - Session, 2-4
 - SystemConfiguration, 2-5
 - UserItem, 2-6
- object structures, 2-2
- object synchronization, 2-11
- parent and child objects, 2-10
- PC requirements, 3-1

structure of a session, 2-4

typical applications, 2-1

creating a new project, 5-1

custom application

creating a new project, 5-1

D

downloading Client API SDK, 3-2

E

examples. *See* programming tips and examples

F

Folder object, 2-7

folders

folder structure, 2-9

how folders are used, 2-7

G

getting help, 4-1

I

installing

- application certificates, 4-5
- Client API SDK, 3-2
- license keys, 4-5
- Simple ViewPoint sample application, 4-3

L

- license requirements, 3-2

N

new project

- creating, 5-1

O

- object group diagrams, 2-3
- object structures, 2-2
- object synchronization, 2-11
- obtaining
 - application certificates and license keys, 4-4

P

- parent and child objects, 2-10
- PC requirements, 3-1
- programming tips and examples
 - How do I change a user's personal status?, 5-11
 - How do I check the availability or personal status of a user (extension)?, 5-10
 - How do I check the status of a Contact Center

agent?, 5-11

- How do I create a call?, 5-5
- How do I create a Wave contact?, 5-9
- How do I export a list of voice messages?, 5-14
- How do I find a specific Call object?, 5-6
- How do I get events for any item?, 5-9
- How do I locate a dialing service by its access code?, 5-15
- How do I log in and star a new session?, 5-5
- How do I record conversations between Contact Center agents and external callers?, 5-15
- How do I sign a contact center agent in and out?, 5-13
- How do I store data with a party in a call?, 5-7

S

- sample application. *See* Simple ViewPoint sample application
- sample code. *See* programming tips and examples
- sample installer program
 - about, 4-6
- Session object, 2-4
- session structure, 2-4
- Simple ViewPoint sample application
 - about, 4-1
 - installing, 4-3
 - reference, A-1
- SystemConfiguration object, 2-5

T

- TeleVantage Client API
 - migrating existing applications to Wave, B-1
- TeleVantage Client API vs. Wave Client API, B-2
- typical applications, 2-1

U

UserItem object, 2-6

W

Wave Client API. *See* Client API

Wave Client API vs. TeleVantage Client API, B-2

