



AOL, LLC

QuickStart Guide

for the

AIM SDK

Revision 1.1
June 2006

Table of Contents

| | |
|---|----|
| AIM SDK Overview | 1 |
| Who Should Read This Guide | 1 |
| What Is In This Guide | 1 |
| Supported Environments | 2 |
| Supported Languages | 2 |
| Installing the AIM SDK | 2 |
| File Directory Structure | 3 |
| AIM SDK Files | 4 |
| Include Files (C/C++) | 4 |
| Library Files (C/C++) | 4 |
| Type Library File (COM languages) | 4 |
| Runtime/Redistributable Files | 4 |
| Key AIM SDK API Conventions | 6 |
| Screen Name | 6 |
| API Key | 6 |
| COM | 6 |
| Strings | 6 |
| Variants | 6 |
| Arrays | 6 |
| Streams | 7 |
| Session Object | 7 |
| Sinking Events | 7 |
| Asynchronous Operations | 7 |
| Transaction IDs | 7 |
| Message Pumps | 7 |
| Threading | 8 |
| Client Information | 8 |
| Properties | 8 |
| Preferences | 8 |
| Preferences Hook | 9 |
| BART | 9 |
| Secondary Sessions | 9 |
| Other Documents and Resources | 10 |

AIM SDK Overview

The AIM SDK provides a standardized programming interface that enables you to integrate America Online's AIM instant messaging functionality into your own software, empowering your user base with a full range of communication options, including presence, text messaging, file transfer, and audio/video communications.

Your users access AIM features seamlessly within your software, while AOL continues to provide a reliable network infrastructure for communications.

The AIM SDK is comprised of APIs that are intended to simplify the process of incorporating AIM features into your software application. Note that this SDK and its APIs focus on network protocol and AIM business logic functionality; graphical user interface (GUI) development tools are not provided.

Who Should Read This Guide

This QuickStart Guide is intended for intermediate to advanced software developers who are ready to use the AIM SDK to integrate AIM functionality into their software.

What Is In This Guide

This guide shows you how to get started quickly with the AIM SDK through the following areas:

- Installing the AIM SDK
- Setting up your development environment
- Understanding the AIM SDK libraries
- Understanding essential AIM API conventions
- Obtaining additional information

Supported Environments

It is recommended that your development environment include:

| Windows Environment | Mac OS X Environment | Linux Environment | Pocket PC Environment |
|-------------------------|----------------------|----------------------------|--|
| Windows 2000 or later | Mac OS X 10.4 | Red Hat Enterprise Linux 4 | Windows 2000 or later |
| Visual Studio .NET 2003 | Xcode Tools 2.2 | gcc 3.4.5 | Visual Studio .NET 2005, Windows Mobile 5.0 SDK for PocketPC |

Supported Languages

On the Windows platform, the AIM SDK supports any COM-compatible language, including C/C++, Visual Basic, and any .NET language. On other platforms, C/C++ is supported.

The AIM SDK also contains **beta** support for Java on Windows, Mac OS X, and Linux, and Objective-C on Mac OS X.

Installing the AIM SDK

The AIM SDK includes APIs and associated libraries that comprise the core programming tools that you will use during development, as well as files to be distributed to users with your software.

To install the AIM SDK:

1. On a local computer to be used to create software using the AIM SDK, create a directory to store the AIM SDK files.
2. Open the AIM SDK zip or tar file and extract the folders and files, retaining the original file structure, into the appropriate directory created in step 1.

Once extracted, your installation of the AIM SDK is complete. However, you must still take steps to incorporate specific files into your development environment and into the product to be distributed to users.

File Directory Structure

Once installed on a local computer, AIM SDK files are accessible via the following directory structure.

| Directory | Description |
|--|---|
| <SDK Root Folder>\devtools\ Example: C:\aimsdk\devtools\ | This directory stores tools useful for developing AIM SDK based applications, including the acchash tool that generates fingerprints. |
| <SDK Root Folder>\doc\ Example: C:\aimsdk\doc\ | This directory contains the documentation for the AIM SDK, including this quickstart guide, detailed API documentation, and several tutorials. |
| <SDK Root Folder>\dist\ Example: C:\aimsdk\dist\ | <p>This directory stores files, including .dll or .dylib files, to be packaged with your software for distribution to users. Under this directory, you will find the following directories:</p> <p>release (srelease) – AIM SDK runtime files to be distributed with AIM custom client software.</p> <p>The base SDK includes just the “release” directory, to support the “Shared” implementation option. Future SDKs may include the “srelease” directory which contains static libraries that can be linked directly into your application, as in the “Static” implementation option.</p> <p>This directory may also contain programs useful for testing purposes.</p> |
| <SDK Root Folder>\include\ Example: C:\aimsdk\include\ | This directory stores the necessary API include files that you will reference within your code. |
| <SDK Root Folder>\lib\ Example: C:\aimsdk\lib\ | This directory stores the necessary API library files that you will link into your code. |
| <SDK Root Folder>\samples\ | This directory has several sample plugins and client applications to demonstrate how to use the API. |

AIM SDK Files

Include Files (C/C++)

The AIM SDK has four primary include files:

- **AccCore.h**: This file brings in the AIM SDK global functions, along with the interface definitions for the AIM SDK objects. All SDK clients will need to use this file.
- **AccDispid.h**: This file brings in the AIM SDK event id definitions; it is only necessary if you are writing a custom event sink.
- **AccSupport.h**: This file brings in the AIM SDK helper classes, used to simplify working with COM and AIM SDK types in C++. On Windows, libraries such as ATL and MFC provide similar functionality, but for cross-platform development, use of this file is recommended.
- **AccVersion.h**: This file contains the version information for the SDK. Its use is completely optional.

Library Files (C/C++)

The AIM SDK has two library files your software may link to:

- **accuiddlib**: This file contains the necessary interface ids (IID_AccXXXX) used when accessing AIM SDK interfaces. Typically, both plugins and custom clients will link with this library.
- **acccore**: This file contains the necessary bindings when calling AIM SDK global functions. Custom clients and some plugins will need to link with this library.

Type Library File (COM languages)

The AIM SDK type information is contained in acccore.dll.

Runtime/Redistributable Files

This section pertains to AIM Custom Clients. Plugin developers need only distribute their plugin and its dependencies.

The AIM SDK is composed of a number of shared libraries, some of which are optional. The base SDK involves just three libraries, but for full functionality, additional libraries for security services and media processing are necessary. For media processing, there are multiple library options.

The following table indicates what libraries must be distributed with your AIM Custom Client application.

| Library | Description |
|--|---|
| xprt5 | AIM SDK Runtime |
| coolcore4x | AIM SDK Protocol |
| acccore | AIM SDK Logic |
| jg*tlk | Optional. Audio software for calls with previously released AOL/AIM products, such as AIM 5.9 and AOL 9.0. If not present, audio calls with these clients will not be supported. |
| sipfoundry sipXtapi 2.9 | Optional. Open source SIP-based audio/video software for calls with newer IM clients, such as AIM Triton, iChat, and other Open AIM clients. If not present, calls with these clients will not be supported. More info: http://www.sipfoundry.org/sipXtapi/ |
| Mozilla NSS 3.9 (or later) (Not included in SDK) | Optional. Security software for SSL AIM sessions, S/MIME encryption of IMs, and other security operations. If not present, security services will not be supported. More info: http://www.mozilla.org/projects/security/pki/nss/ |

Currently the AIM SDK libraries are provided as dynamic libraries (.dll/.dylib). Future versions of the SDK may include the core AIM SDK libraries as static libraries (.lib/.a)

Key AIM SDK API Conventions

Screen Name

A screen name is what identifies a user on the AOL IM networks, and is necessary to register for an API Key and to test the AIM functionality that you are developing. If you do not have a screen name, you can obtain one from www.aim.com.

API Key

In order to develop an AIM Plugin or AIM Custom Client, you will need to get a key from developer.aim.com. This key is used as the UUID for AIM Plugins, and in the client description string for AIM Custom Clients.

COM

On Microsoft platforms (Windows and PocketPC) the AIM SDK APIs are exposed in the form of COM interfaces. The typical COM rules and conventions apply. On other platforms, the AIM SDK provides an infrastructure similar to COM via the `AccSupport.h` header file.

Strings

To support international character sets, strings in the AIM SDK are in (2-byte) Unicode format. Input parameters may be null-terminated Unicode strings or standard COM BSTRs. Output parameters are always BSTRs which must be deallocated by the caller using the COM **SysFreeString** API or the AIM **AccStringFree** API. The `AccSupport.h` header file includes a **CAccBstr** helper class to help manage BSTR objects.

Variants

The AIM SDK frequently makes use of variant data structures to hold data that can have many possible types. These structures are standard COM VARIANT structures. Before using the data in a supplied variant, you should check that the type "vt" of the variant matches what you expect. Otherwise, you may treat the data incorrectly. When variants are returned as output parameters, they must be deallocated by the caller using the COM **VariantClear** API or the AIM **AccVarClear** API. The `AccSupport.h` header file includes a **CAccVariant** helper class to help manage VARIANT objects.

Arrays

Lists of data are passed to and from the AIM SDK using standard COM SAFEARRAYs, stored inside a VARIANT. Typically the SAFEARRAYs are of type VT_VARIANT, except when accessing "data" properties, in which case the type is VT_UI1. When SAFEARRAYs containing VARIANTS are used, the interface documentation will indicate what types are used in the VARIANTS. The `AccSupport.h` header file includes a **CAccArray** class to help manage COM SAFEARRAY objects.

Streams

When supporting SAFEARRAYs of bytes (VT_UI1), the AIM SDK also supports methods to get the binary data as a stream. For example, **IAccBartItem** provides properties to get the item data as an array (**AccBartItemProp_Data**), or a stream (**AccBartItemProp_DataStream**). These streams support the standard COM **IStream** and **ISequentialStream** interfaces. They also support the **IAccStream** interface, which allows for the installation of a callback to get asynchronous stream notifications. By passing in a callback object to **IAccStream::SetAsyncListener**, your callback will be notified when data is available to be read on the stream, or the stream is closed.

Session Object

The core interface in the AIM SDK is the **IAccSession** interface. This interface abstracts a session with the AIM network. If you are creating a custom AIM client, you will need to create an instance of this object with the **AccCreateSession** function. If you are creating an AIM plugin, the **IAccSession** interface used by the AIM client will be passed to you via your plugin's **IAccPlugin::Init** method.

Sinking Events

Once you have an instance of the **IAccSession** interface, you will generally want to register to receive AIM SDK events from the session. This is accomplished in typical COM fashion by connecting to the **DAccEvents** connection point. The **AccSupport.h** header file includes an **CAccEventSink** helper class that you can derive from to simplify the process of registering for and sinking events. ATL and MFC also have mechanisms that can be used to sink events.

Asynchronous Operations

The AIM SDK is nonblocking, meaning that its APIs always return immediately, and often return their status through a **DAccEvents** callback. For example, **IAccSession::SignOn** indicates its status via **DAccEvents::OnStateChange**; callbacks will fire for the various intermediate states until a final online or offline state is reached. Request-reply APIs, such as **IAccSession::LookupUsers**, result in a single asynchronous callback, such as **DAccEvents::OnLookupUsersResult**. The API documentation indicates which callbacks correspond with which API calls.

Transaction IDs

Request-reply APIs require context so that the reply callback can be matched to the original request. The AIM SDK uses the concept of transaction ids to do this matching. A call such as **IAccSession::LookupUsers** returns a **AccTransId** identifier; this same identifier is passed as an argument in the **DAccEvents::OnLookupUsersResult** callback. It is very important that an API client keep track of its outstanding transaction ids and only handle callbacks that contain ids that it knows about. Otherwise, unexpected behavior will occur when a callback is processed by a client that did not initiate the corresponding request.

Message Pumps

In order to ensure that callbacks are received, the application must pump messages. For plugins, this is automatically handled by the host application. However, if you are writing a custom AIM client, this means that you must run a message loop, either by calling **AccMessageLoop**, regularly calling **AccDispatchMessages**, or on Windows, running a standard Win32 message loop. Since the entire SDK is event driven, clients of the SDK must

not block in any callbacks; doing so will stall the message pump and prevent further events from occurring.

Threading

The AIM SDK is level-1 thread-safe. This means that the global APIs (e.g. **AccCreateSession**) can be safely used from multiple threads, but individual objects are not thread-safe and cannot be accessed concurrently. Objects follow the COM “apartment” model, which means that objects can only be accessed from the thread on which they were created.

Client Information

If you are creating a custom AIM client, it is important to identify your client to the AIM network, so that the network can identify who is using it and protect itself against malicious clients without interfering with the operation of normal ones. To identify your client, access the **ClientInfo** property on the **IAccSession** interface, by calling **IAccSession::get_ClientInfo**. This will return an **IAccClientInfo** object, which you can fill in with your client’s information via the **IAccClientInfo::put_Property** API. Most of the properties are optional; only the **AccClientInfoProp_Description** property must be filled in. The description can be of your own choosing as long as it includes the client key that has been assigned to you; the suggested format is “MyClientName (key=a23b1db3478dahd)”

Properties

Almost all AIM SDK interfaces support **get_Property** and **put_Property** calls that take an enumerated type and a VARIANT as arguments. This allows us to add additional properties without breaking compatibility with API clients who are not expecting the new property. For each property enumeration, the API documentation specifies what type will be returned from a **get_Property**, and what types, if any, are acceptable for a **put_Property**.

Some interfaces also support a **RequestProperty** call, which asynchronously retrieves the value of a property that may not be cached on the local machine; an example is **AccUserProp_AwayMessage**. Calls to **RequestProperty** will result in a callback (specified in the documentation) from the **DAccEvents** interface with the property value.

Preferences

The AIM SDK has many internal settings that control behavior of the SDK. An example is the user’s saved AIM profile, which is stored under the AIM SDK preference specifier “aimcc.general.profile”. These settings are accessed through the **IAccPreferences** interface, where the **GetValue** and **PutValue** calls are used to read and write preferences. A complete list of preference specifiers used by the AIM SDK is provided in doc/PrefsSpecifiersSimplified.txt

Plugins can read and write their own preferences to the AIM preferences store by supplying their own specifiers, e.g. “myplugin.textColor”. These specifiers must start with a prefix other than “aimcc”.

Sometimes the AIM SDK provides a way to set data either through an API **put_Property** call, or through a preferences **PutValue** call; one example is the setting of a buddy icon. In cases like these, the distinction is that a value set through **put_Property** will only last for the duration of the session, whereas the value set through preferences value persists permanently.

Preferences Hook

If you are writing a custom AIM client, you will need to supply a preferences hook to **IAccSession::put_PrefsHook**; this hook will be called when AIM SDK needs to read or write a preference from a persistent store. Your hook is responsible for reading or writing the preference value to the persistent store. It can also “force” preferences to a given setting by always returning a specific value; an example would be to always return “-1” for “aimcc.connect.secure” to force the AIM connection to always use SSL.

BART

BART (an acronym for Buddy ART) is a general name for metadata associated with a user, including the user’s buddy icons, sounds, wallpaper, status text, and other user-specific items. The interface used for dealing with BART items is the **IAccBartItem** interface, which is returned from **IAccSession::get_Property** and **IAccUser::get_Property** for the buddy icon and other aforementioned properties. The **IAccBartItem** interface provides additional information about the data, including whether it is AOL-sanctioned (**AccBartItemProp_Official**), and its MIME type (**AccBartItemProp_MimeType**). The item data is accessed via **AccBartItemProp_Data**, but because the data for a given BART item may not be already cached on the local machine, it is recommended that **RequestProperty** be used when attempting to retrieve the item data.

Secondary Sessions

The **IAccSession** interface abstracts a session with the AIM network, and provides methods such as **SignOn** and **SignOff** to control interaction with the AIM network. The **IAccSecondarySession** interface abstracts a session with one or more AIM users, such as an IM session or a audio-video session. **IAccSecondarySession** provides basic session disposition and control methods, such as **Accept**, **Reject**, and **Invite**; each type of session also has a specific interface that derives from **IAccSecondarySession** to provide service-specific methods; an example is **IAccImSession**, with methods such as **SendIm**. When you get an **IAccSecondarySession** interface, you can check its **ServiceId** property to determine what type of session it is; you can then **QueryInterface** for the specific interface (e.g. **IAccImSession**) for that session type.

Secondary sessions can be created from **IAccSession::CreateImSession**, **IAccSession::CreateChatSession**, or from methods on the secondary managers (**IAccAvManager**, **IAccFileXferManager**, etc.) Secondary managers can be accessed by calling **IAccSession::GetSecondaryManager** with the appropriate service id.

Other Documents and Resources

This document provides a high-level overview of the AIM SDK. Depending on whether you are building an AIM plugin or AIM custom client, there are other documents and sample code that will help get you started.

Plugins

For plugins, first consult the Plugin Tutorial documents; the C++ tutorial is under `doc/tutorials/plugin_cppatl`, and the C# tutorial is under `doc/tutorials/plugin_csharp`. You can then look at the code presented in the tutorial under `samples/plugins/myatlplugin` and `samples/plugins/mycsplugin`.

More complex functionality is demonstrated in the **JAMS** sample plugins included in the SDK under `samples/plugins/jams`. The following AIM SDK features are demonstrated by the **JAMS** plugins mentioned below:

| | |
|-----------------------------|--------------------------------|
| Accessing Buddy List | FightClub, BuddyListOps |
| Setting away message | LWAWay |
| Setting profile | MyTunes |
| Setting buddy icon | IconMaker |
| Logging IM conversations | EZLogger |
| Responding to incoming lms | MobileControl |
| Pre-processing received lms | PhoneNumToLink |
| Post-processing sent lms | Colorizer, EZFormat |

Custom Clients

For custom clients, consult the Custom Client Tutorial document under `doc/tutorials/client_cpp`. You can then look at the code presented in the tutorial under `samples/accsample`.

Other simple clients in the samples directory include **aatlbuddy** (a C++/ATL client), **avbbuddy** (a VB.NET client), **acshbuddy** (a C# client), **accjsample** (a Java client), and **pocketsample** (a C++ Pocket PC client).

Use of the AIM SDK Location APIs is demonstrated in the **acshlocation** C# sample client. The concepts shown in this sample are easily implemented in other languages.

Complex functionality, including multi-user chat and audio/video is demonstrated in the **amfcbuddy** graphical MFC-based sample client under `samples/amfcbuddy`.

Bots

AIM Bot sample code is provided in **accbot**, a cross-platform C++ bot, and **accjbot**, a Java bot. **accbot** and **accjbot** are both readily customizable for your specific purposes.

Other Documents

Searchable, detailed documentation about each interface and method is available in the `doc/accdocs.chm` file. Java developers can also access the javadoc documentation for the AIM SDK under `doc/accjwrap`. Detailed information on specific topics can be found under `doc/technotes`.