Using the Skype Client as a Server Platform

A Connectotel White Paper

Prepared by
Marcus Williamson
Connectotel
London, UK
http://www.connectotel.com/
marcus@connectotel.com

Created: 1 March 2005
Last Edited: 8 May 2005

Contents

1.0 Introduction

Skype offers a rich Application Programming Interface (API) for the Skype client software. The “Skype Access API” provides developers with the ability to process Voice over Internet Protocol (VoIP) calls and to intercept Instant Messaging (IM) chat messages, as well as being able to query the Skype user directory and locally-held Contacts list.

The architecture model for application development most often used so far has been the “client-side” model, in which the application software is downloaded and installed by the end-user to operate in conjunction with the user’s Skype client.

Another model, less frequently used, is a “server-side” model where the application runs on a machine somewhere within the Skype network. Until recently the only service which used this model was the Skype “test123” service, which echoes back a message to the user. The recently introduced ‘SMS to Skype’ and ‘Skype to SMS’ services, offered by Connectotel, are an example of this model, providing added functionality via a virtual user which has the Skypename “smsgateway” within the Skype network.

For certain applications a hybrid “client/server” model may be used, where part of the application runs “client-side” and the other “server-side”.

The following document is intended for application designers and developers who wish to create services running as virtual users on the Skype network, running as either “server-side” or “client/server” applications.

This paper will not discuss in detail the design or development of the application written using the Skype API, as this subject is already covered in the Skype API documentation and online resources.

2.0 Architecture Models

2.1 Client-side

In the first few months of existence of the Skype API, developers for the Skype platform have focussed their efforts on creating add-in applications which will run on the end-user’s computer system, interacting with the Skype client there. These applications typically provide some supplemental functionality to the Skype client, for example in the form of an answering machine or call-forwarding capability.

When using this type of software product, the user is required to download a software package and install it on their own computer. Once installed, the software provides its functionality by interacting with the Skype client software, intercepting voice calls and/or instant messages, or interacting with the Skype user directory, depending on the application.

The advantage of this model is that the software is deployed directly onto the user’s client machine and uses the resources of that machine to carry out all processing. This in turn could also be seen as a disadvantage, in that the application developer is potentially supporting the entire community of users who are using the application, with consequent

demands on time and resources for the developer. When an application such as this requires updating to a later version, the user is required to download and reinstall the software.

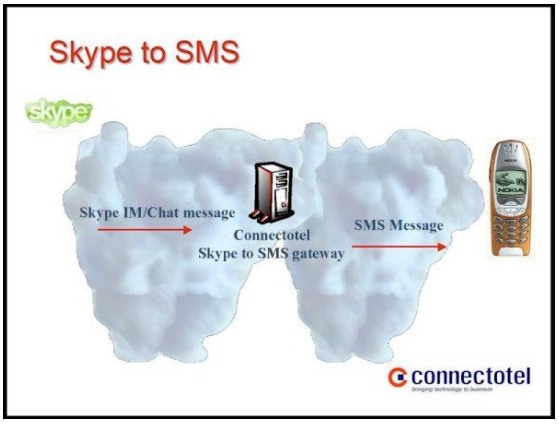
This application architecture we shall call “client-side” for the purposes of this paper.

2.2 Server-side

A second model for developing applications is what we shall call here a “server-side” model.

In this case the “user” is a machine which is providing some specialised functionality to other users within the Skype peer-to-peer network, such as a gateway to an external non-Skype instant messaging system, or a link to an external data feed.

When using this type of service, the software developed typically runs on only one “server” machine within the entire Skype network and does not require any software to be installed onto the end-user’s machine. In some cases the workload may be shared between two or more machines. For example, in the case of incoming and outgoing SMS messages, one machine might be configured to handle the incoming traffic and the other to handle outgoing traffic. The user will perceive one “service”, but that service is being provided by two “server” machines within the Skype network.



The user would communicate with one machine, named for example “skype-to-sms”, to send SMS messages and the other machine, named for example “sms-to-skype”, would be responsible for handling the traffic originating as an SMS message.

When the user wishes to use the service provided by the application, he/she initiates a voice call or starts an instant message chat session with the “server” machine. The machine then carries out the necessary action and provides responses generated by a program running in conjunction with the Skype client, written using the Skype API.

The advantage of the “server-side” model is that the application exists in just one location. There is no software required on the client and therefore less of a resource requirement for support and maintenance of the software. In the event that modifications are required to the application, these are carried out in one place, on the server machine.

2.3 Client/Server

The third model, which we shall call “client/server”, is where part of the application is running “server-side” and another part runs “client-side”. In this model, the application’s intelligence is shared between the front-end and back-end, with both components cooperating to achieve the required task.

The client/server model provides a means to add functionality alongside the existing Skype user interface. For example, in the ‘Skype to SMS’ application already cited, a possible

enhancement would be to add an application at the client side which allowed the user to choose a target user, from the Skype directory, then type a message for transmission to the target user's mobile phone. The message would then be passed via the Skype chat functionality to the backend server for processing. No applications of this type yet exist for Skype.

This model has the advantage of the "server-side" model, in that the main backend application exists in just one location. However, in also having a "client-side" software component, the developer will be subject to the disadvantages of that model in terms of potentially supporting and maintaining software on a large number of end-user machines. This model also raises questions of version control, where the front-end and back-end components must decide on a common command set to use. This issue can be resolved relatively straightforwardly by carrying out a version-checking handshake process when the "client-side" and "server-side" components first start to communicate.

3.0 The Skype client as a "server"

3.1 Server concepts

When using the Skype client as a server for an application, the developer will use the Skype Access API in the normal way, just as he/she would do with a "client-side" application. The calls and/or chat messages will arrive from Skype users, who may be located anywhere within the Skype network.

As the call or chat message arrives, the server may verify the user's status (see section 4.0 below) then carry out the necessary processing of the incoming data. After completion of processing, the server will hand any results back to the requesting client.

As a simple example, let us consider a server which has been developed to provide currency exchange rate information. In this application the data flow would be:

- User starts a chat session with the "exchange rate" server and types the currency symbol required, for example USD (US Dollar).
- The "exchange rate" server receives the chat message containing "USD"
- The "exchange rate" server looks up the current currency exchange rates for the US Dollar from a file received via an external data feed.
- The "exchange rate" server sends back the results to the requesting user as a chat message.

The developer can, of course, incorporate additional intelligence into the software, such as replying to the user in their own language, based on the user's profile language, or tailoring the information in the reply to any information in the user's profile such as their country, gender or birthdate.

It is recommended for applications using the Skype chat facility, which do not have a client-side portion, that the developer incorporates a HELP command, which can be used to inform the user about how to interact with the server. The HELP command would summarise the commands available, together with the syntax for those commands.

3.2 Configuration

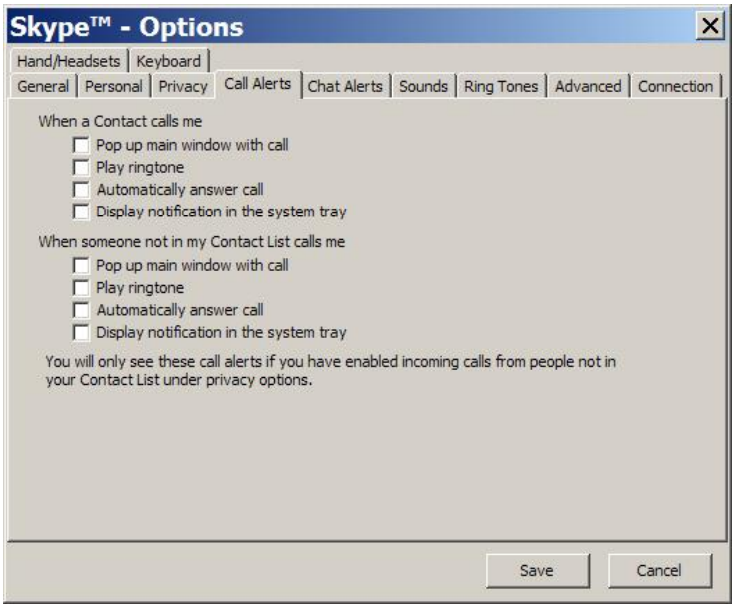
The Skype client is designed to provide a user-friendly interface to the Skype transport for VoIP and IM services.

For an ordinary user, the interface provides ongoing information about the status of calls, chats and the online and authorization status of Contacts. However, when used as a server there are a number of notifications, pop-ups and alerts which may interfere with server operation, reduce performance and clutter the screen. These are:

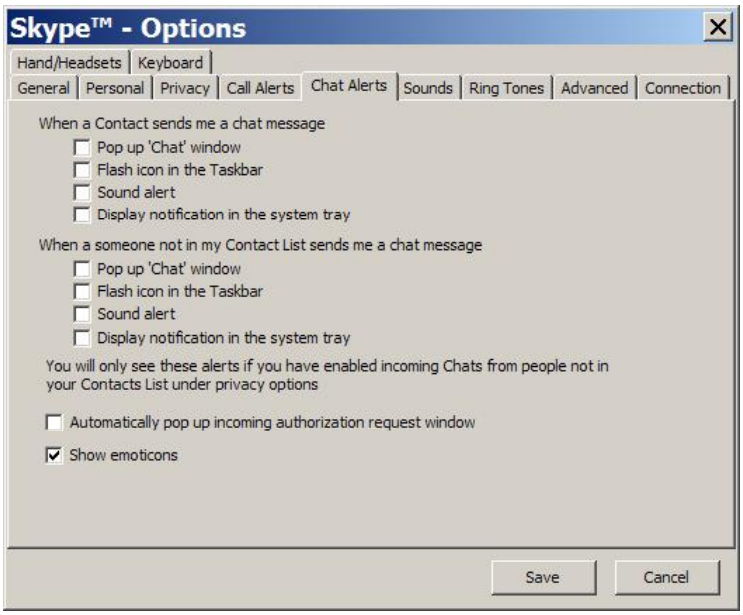
- Call and Chat message notifications
- Authentication Requests
- Sounds
- 'Advanced' Options

3.2.1 Call and Chat message notifications

The following screenshot from **Options / Call Alerts** shows how to disable unnecessary pop-ups and sounds, for server operation, when receiving calls. All checkboxes in this case are unticked, so as to disable pop-up windows, sounds, automatic answering and notification in the system tray.



The following screenshot from **Options / Chat Alerts** shows how to disable unnecessary pop-ups and sounds, for server operation, when receiving chat messages. Again, all checkboxes are unticked, so as to prevent the chat window and taskbar icon from popping up and disable the sound alert and the notification in the system tray.

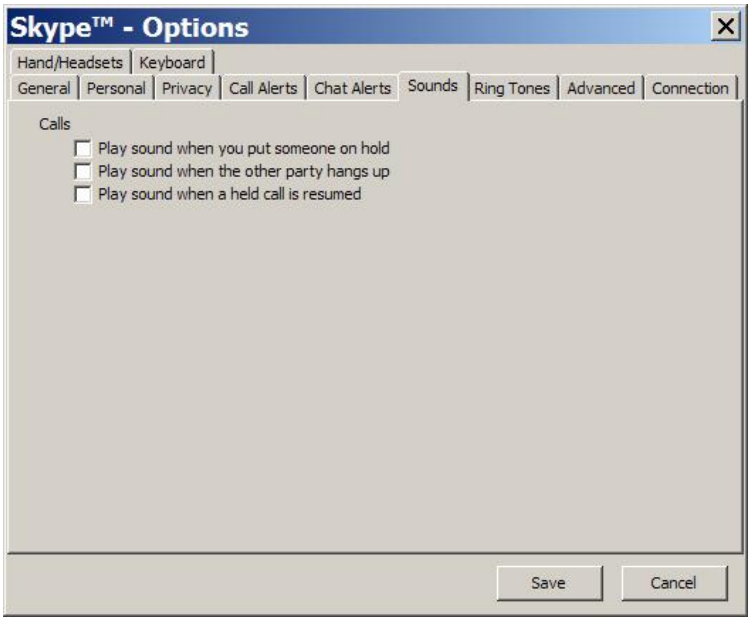


3.2.2 Authorization Requests

The screenshot above also shows how to disable the pop-up for an incoming authentication request, by unticking the checkbox “Automatically pop up incoming authentication request window”. When this checkbox is unticked, the authentication request will not prompt the user but will instead be saved as an “Authorization Waiting” which is displayed in the Skype client “Start” window. The concept of Authorization is discussed further in section 4.2 below.

3.2.3 Sounds

Disabling sounds can reduce the processor utilization on systems intended for “server” use. This screenshot shows all three “play sound” notification checkboxes unticked to disable the sound in each case:

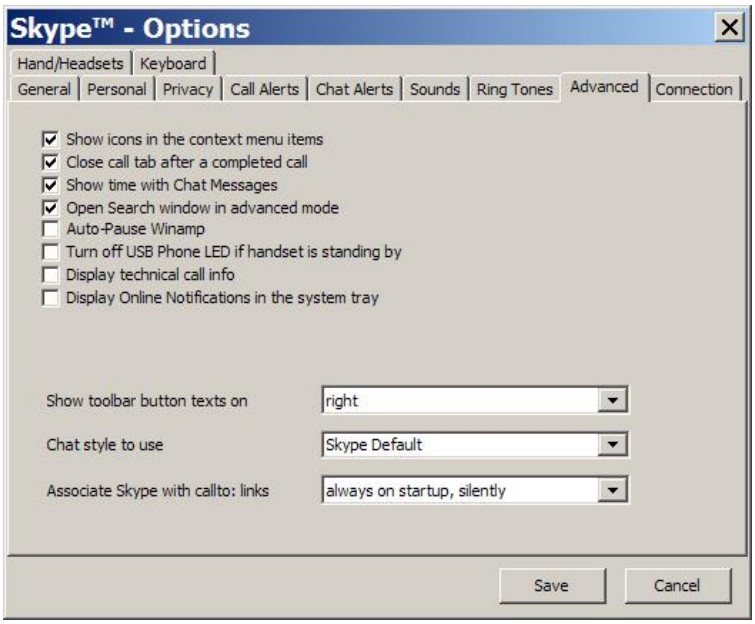


3.2.4 'Advanced' Options

Within the Advanced options there are two settings which can help reduce the load on "server" systems. These are:

"Close call tab after a completed call" – This should be ticked
and
"Display Online Notifications in the system tray" – This should be unticked

Both these settings are shown in the screenshot here:



4.0 Access Control Mechanisms

The following access control mechanisms are available to the application developer who wishes to make use of the native Skype functionality for access control on a "server-side" system:

- Contacts List
- Authorization
- Blocked Users

Alternatively, the developer may choose to use his/her own mechanism for providing access control.

4.1 Contacts List

The Skype Contacts list is also known as the "Friends" list or "Buddy" list. On a conventional "client" configuration, this is list of contacts with whom the user may communicate regularly. In a server environment, the administrator adds entries into the Contacts list by using the menu option **Tools / Add a Contact** or by adding the user into the Contacts list when prompted to "authorize this user".

To be entered onto the Contacts list, a user should request authorization from the Skypename which is running the “server” application. Once the administrator receives the authorization request he/she may choose whether or not to add the user to the list of authorized user’s for this Skypename. If the administrator accepts the authorization request then the user will be added to the Contacts list for the server.

When using the Skype client in a server configuration, the Skype API can be used to determine whether a user is in the Contacts list of the server machine by testing the user property BUDDYSTATUS using the command:

```
GET USER username BUDDYSTATUS
```

The possible return values for this command are:

0 The user has never been in the Contacts list
1 The user was in the Contacts list but has been deleted
2 The user is pending authorization
3 The user is in the Contacts list

(See section 7.8.3 of the Skype API documentation)

The “Contacts List” option can be used to build up a limited list of Alpha Test or Beta Test users who are permitted to use the service, by ensuring that only those users who have “BUDDYSTATUS 3” be granted access. In this way, the developer can maintain control of who will use the application for the duration of the testing cycle.

4.2 Authorization

The Skype API can be used to determine whether a user is authorized on the server machine by testing the user property ISAUTHORIZED using the command:

```
GET USER username ISAUTHORIZED
```

(See section 7.8.3 of the Skype API documentation)

This command returns either TRUE or FALSE depending on whether the user is authorized. Being “authorized” means that the user is able to see the online status of the virtual user with which the server is associated.

The “Authorization” option can be used on its own, or in conjunction with the Contacts list, to control access to a server machine.

Note: It is important to be aware that granting authorization within the Skype client is currently a **manual process** for which no APIs exist. This implies a potentially large amount of work for personnel who will have to grant or deny authorization one-by-one to the users who request it.

4.3 Blocked Users

The ability to block users can be useful in the event that one or more users attempts to abuse a server facility. The Skype client allows maintenance of a list of blocked users via the menu option **Tools / Manage Blocked Users**.

Once added to the Blocked Users list, communications from the blocked user will no longer be seen by the Skype Client, nor indeed by the Skype API. The block is absolute, but reversible in the event that the user should be again allowed to communicate with the server.

The Skype API can be used to determine whether a user is blocked from accessing the server machine by testing the user property ISBLOCKED using the command:

```
GET USER username ISBLOCKED
```

(See section 7.8.3 of the Skype API documentation)

This returns either TRUE or FALSE depending on whether the user is blocked.

4.4 External Access Control

As well as the methods indicated above, which are built into the Skype client, the developer may decide to implement his/her own external access control mechanisms. This would typically involve building a list of known allowed users of the service, then comparing the Skypename of the incoming user with the list of allowed users and granting or denying access accordingly.

Specialised access control mechanisms might involve allowing access based on a user's geographical location and/or language. In this case the developer can make use of the user's profile information to determine these parameters.

5.0 Status Information

It will be important for organisations offering a service on the Skype network to provide a means for local and/or remote monitoring of the server system. There are a number of ways in which this could be carried out including:

- Logging
- Status Commands
- Remote Access

5.1 Logging

Logging will provide the developer with historical information about the status of the server application and its interaction with users. When building a server application for Skype, it may be useful to log to a file all Skype API messages sent and received to assist with diagnosis of any problems which may occur. If this is done, it is recommended that the Skype API LASTONLINETIMESTAMP messages are omitted from the log, unless these are

required by the application. Doing this will considerably reduce the amount of data which will be stored in the log. An application developer may choose to filter other, non-essential messages in a similar way.

5.2 Status Commands

The developer may choose to incorporate status commands into his/her application. This would return administrator statistics, or other data, in response to a given command. For example, in the currency "exchange rate" server mentioned in section 3.1, the developer might implement a command STATUS which returned:

```
Server up since: 1 April 2004, 12:35:33
Current date/time: 28 February 2005, 18:34:23
Last data feed received: 28 February, 18:30:18
Number of currencies recognised: 88
Number of requests processed: 6745
```

The use of the STATUS command could be restricted for use only by those users who are in the server's Contacts List, in which case the Contacts List on the "server" system could be used to store a list of recognised administrators for the server.

An alternative to providing status information on demand in this way is to write status information to a web page on a web server. This status page can then be queried by a remote administrator when required, using a web browser.

5.3 Remote Access

If the server is located outside of the developer's normal office environment, for example at a hosting facility, then it may be necessary to provide remote access to the server machine. The Skype client cooperates well with remote access solutions such as PCAnywhere and VNC. Using this type of software the developer can interact with the application and the Skype client exactly as if he/she were sitting next to the server machine.

6.0 Fault Tolerance

6.1 Detecting the presence of the Skype client

An application can determine whether the Skype client is present and connected to the Skype network by issuing the command PING, to which the response, if all is well, is PONG.

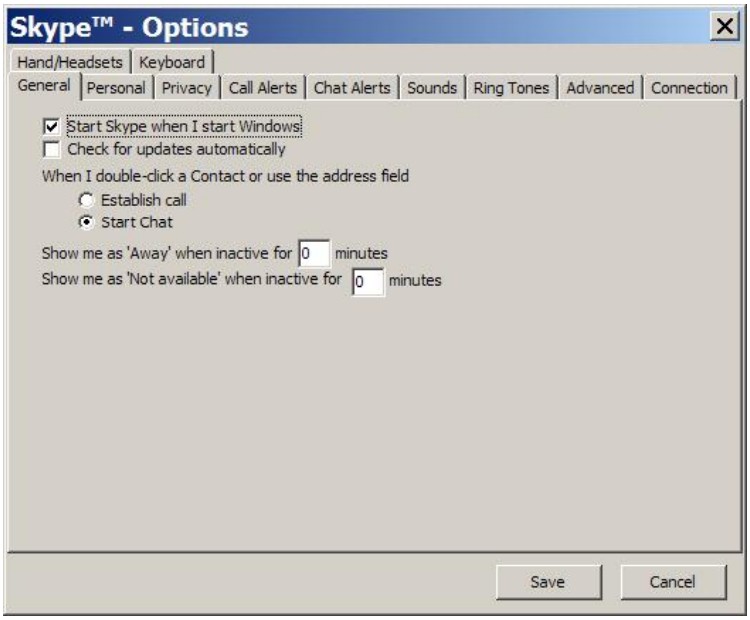
(See section 7.9.8 of the Skype API documentation)

In the event that the Skype client is not present, an application can launch the Skype client using the ShellExecute function, or similar, depending on the programming language being used.

6.2 Automating start-up

It is recommended that the Skype client and server-side software be made to start automatically in case a server reboots spontaneously or has been powered off inadvertantly and restarts. The server-side software can be started automatically if it is running as a Windows service, or via a shortcut in the Windows Startup folder.

The Skype client software can be started automatically by ticking the checkbox shown in the screenshot below. It is recommended that the option “Check for updates automatically” be unticked so that the Skype client startup process is not interrupted by a prompt to update the Skype client software. If an update to the Skype software is required, this can be managed by the administrator manually.



6.3 Standby server

In the event that a server system experiences a hardware or software failure, it will be useful to have a standby server available as a replacement. This server should be configured as similarly as possible to the production system, including the Skype client software, application software and configuration options shown in section 3.0 above.

This machine could be a “warm” standby, in which case it already exists on the network under another Skypename, but is not normally available. Alternatively, as a “cold” standby, it is a machine which is normally powered off but is a “clone” of the production server.

When using a “warm” standby, manual intervention will be required to bring the new server online. This will involve logging out the existing “server” machine under its normal Skypename and then using that same name to log in the standby machine.

A “warm” standby, using a different Skypename, could also be brought in as a production server in the event of routine maintenance on the usual production server.

7.0 Legal

If using Skype as a server, please make contact with Skype to determine whether doing so is within the terms of the End-User License Agreement (EULA).

Screenshots are taken from Skype version 1.1. The appearance of the Skype version you are using may be different.

8.0 Contacts

Marcus Williamson, Connectotel
http://www.connectotel.com/
E-mail: marcus@connectotel.com