Creating ICQ applications using icqlib*

Zvika Brakerski† Asaf Koren‡

May 9, 2001

About this document

This manual is submitted as a part of the csicq[1] project documentation.

When working on the csicq project, we have been using a great deal of icqlib. The lack of documentation for the package has been, therefore, a major obstacle in the development process.

However, since we figured out the basic concepts on which the package based, it seemed very intuitive and easy to work with.

Therefore, as a part of our project documentation, we decided to add a manual for the basic features of icqlib **as we figured them out**.

We should emphasize that this documentation is submitted with no warranty what-so-ever. All information here is based on our explorations and cannot be trusted to be true in any way. We take no responsibility on any action taken or not taken based on this manual and hereby declare that to the best of our knowledge, all information in this document might be completely wrong.

The writers of this manual are not affiliated in any way with the writers and/or maintainers of the icqlib package.

We should also make it very clear that this document refers to version 1.0.0 of icqlib. We have not examined other versions.

---

*http://www.kicq.org/icqlib.shtml

†zvika@eng.tau.ac.il

‡asaf@math.tau.ac.il

[1]http://csicq.sourceforge.net

Contents

1 How to read this manual

Throughout the document, we refer to the icq.h file which is the main header file for the icqlib package. Sometimes we will quote the relevant parts of the file where in other times we may only point out the line numbers of the code we're talking about. Therefore if you want to get the best out of this manual, you should have a copy of the icq.h file around when reading this.

2 General structure of the icqlib code

The functionality of the icqlib could be roughly divided into two major parts:

Passive functions (handlers, callbacks). Handlers are invoked by the network[2] in response to certain events.

Lets say someone sent you an ICQ message. How can you tell that this has happened? Well, the ICQ network will send you packets describing the message (the beauty of using icqlib is that you do not need to know how these packets are composed nor how they are interpreted). The icqlib code will listen on the network (we will explain how to we make it listen later on) and upon receiving and interpreting those packets, it will invoke the correct handler. In this example this would be the `icq_RecvMessage` handler (line 114).

But what is the functionality of the handler itself? Now **that** is up to you. It is up to the programmer to program the handlers and assign them to the correct place in the `ICQLINK` struct (we will elaborate on that a great deal later).

Active functions. Okay, so we know (in general terms) what happens when the network informs you of something. But what if you want to notify the ICQ network of some event? Say you want to send a message to a friend?

Well, what you do is just call the `icq_SendMessage` (line 234) function with the appropriate arguments and icqlib will wrap your message and send them to the ICQ network in the correct format.

[2] Of course this is only an abstraction, the network does not really "invoke" any event in the code. What really happens is that the icqlib code listens on the network and whenever the network sends relevant information, the handler is invoked. However, for our purposes, such a description would suffice.

Other than things you might **want** to do, there are also things you **have** to do in order to be connected to the ICQ network. This includes connecting to the server and logging in to the network (of course) and letting the network know you are still connected (namely keepalive). There are functions for all of these (and more) which will be discussed further along this document.

In the next section we will see exactly how these two types of functions are implemented in the code.

3 Detailed examination of the structure of the icqlib code

3.1 The `ICQLINK` struct

The `ICQLINK` type is a type definition of `struct icq_link`. This long structure is defined in lines 69–207 of the code. Its one purpose is to hold information on a single ICQ session.

Let's take a close look at the struct:

General (lines 71–78): In this sections you will find variables that hold general information about the connection such as the UIN of the user, its nickname and password, the local IP and port and the ICQ status of the user.

You should refer to this information as "read only". If you need your current UIN or need to know your status, you can use them but unless you **really** know what you're doing, don't try to change them by yourselves. This is why you have icqlib, to manage this information correctly without you having to worry about parsing network response and things like that.

UDP (lines 81–86) and TCP (lines 89–96): As you know, the ICQ network is composed of all sorts of UDP and TCP links. These two sections hold information that is required for the icqlib in order to maintain these connections. This includes mainly socket file descriptors and related structures.

Again, do not touch unless you know what you're doing. I don't see a reason why you would even want to **read** this information for most basic purposes.

Proxy (lines 99–109): Well, naturally you'll find the proxy settings here. Since we did not use the proxy feature in the csicq project, we prefer not to review it in this document.

Callbacks (lines 112–206): This is a set of pointers to functions[3] to which you assign the callbacks.

Unlike the previous sections, you would usually refer to these as "write only". After initializing your connection, you will assign your callbacks to where they belong and just forget about it for the rest of the session.

The callbacks determine how your connection reacts to events. In the previous section we described the `icq_RecvMessage` callback. Now let's take a different example: `icq_UserOnline` (line 150). Naturally when programming a client, you would like to notify the user when a member of the contact list comes online. Say you have some graphic front-end and you'd like to paint the name of that user in blue (just an example).

The prototype of `icq_UserOnline` is:

```
void (*icq_UserOnline)(struct icq_link *link, unsigned
long uin, unsigned long status, unsigned long ip, unsigned
short port, unsigned long real_ip, unsigned char tcp_flag
);
```

Now what we do is first write a function that does what we want to do (paint the name of the user in blue):

```
 void my_UserOnline(struct icq_link *link, unsigned
long uin, unsigned long status, unsigned long ip, unsigned
short port, unsigned long real_ip, unsigned char tcp_flag
)
{
paint_it_in_blue(uin);
```

[3] If you don't remember how you point to a function in **C**, please refer to your favorite textbook. This is essential in order to understand the icqlib architecture and we will not cover it here.

```
/* Or just do whatever you want to do when a user goes
online.  */
return;
}
```

And then all you have to do is "plug it in" after the initialization of the connection (we will elaborate on the initialization process later):

```
link.icq_UserOnline = &my_UserOnline;
```

And from that moment, whenever icqlib identifies a message from the ICQ network saying some user from your contact list went online, your callback function (`my_UserOnline` in our example) will be called.

3.2 The "active" functions

These functions can be found after line 213. These functions, as explained before, provide an interface for the ICQ network.

There is actually not much to explain about these functions other than discussing each of them specifically. We should just mention that for most of these functions you need to provide an `ICQLINK`. This link in fact identifies the connection on which you want to operate so this is important you provide the correct link (this is important when working on a multi-user client).

3.3 The function `icq_Main`

This function which can be found in line 227 might look like a regular "active" function but in fact it has a very special role when writing an icqlib based ICQ client.

The `icq_Main` function takes as input the `ICQLINK` of a connection and checks in a *non blocking* manner if there is anything new to read from the network. If there is, this function calls all the relevant callback functions and then return.

This function is extremely useful because it releases the programmer from responsibility to the internal flow of control inside the icqlib component. In fact it is this function that enables the abstraction of an event-driven mechanism in icqlib.

4 An ICQ session with icqlib — step by step

Now that we know the basics, we can go and really program an ICQ client. We will now go step by step and find out what we need to do in order to have a successful ICQ session using icqlib.

4.1 Initializing the `ICQLINK`

First of all, we should get ourselves some session identifier, we do that by simply declaring a variable of that type. You could use dynamic allocation if you prefer.

After doing that, we need to initialize it, meaning clean all the garbage and put good initial values in the structure. This is done by calling `icq_Init` (line 217). Note that the function requires a *pointer* to a previously allocated `ICQLINK`. Other than that, you'd require your ICQ UIN, password and nickname.

So here is what we do:

```
ICQLINK link;

icq_Init(&link, 103199388, ''topsecret'', ''tester'');
```

You should also be aware that every initialized `ICQLINK` needs to be "freed" using `icq_Done` (line 219) when you're done with it. We will discuss this later.

We got ourselves an `ICQLINK`, now let's connect to the server.

4.2 Assigning callbacks

Well, now you are actually *writing* your client. Just write the callback functions you would like to use (not all of them have to be assigned) and assign them to the right place in the `ICQLINK` as explained before.

4.3 Logging into the ICQ server

The connection involves two parts. First you connect to the ICQ server and then you login to the network using your UIN and password.

To connect to the server we use `icq_Connect` (line 220) and to log in we use `icq_Login` (line 229).

The arguments for the connect function are quite obvious (you need an address and a port number of an ICQ server). For the login function, you need the status you wish to login to. You can find the possible options for status in `#define` statements in lines 32–39.

so now we do:

```
 icq_Connect(&link, ''icq.mirabilis.com'', 4000);
icq_Login(&link, STATUS_ONLINE);
```

4.4 Staying alive with `icq_KeepAlive`

Now that we're online, we'd like to stay that way. We wouldn't want the ICQ server to disconnect us just because we're idling.

In order to notify the ICQ server that you're alive, just use the `icq_KeepAlive` function (line 228). If the ICQ server does not get a keep-alive sign in two minutes, you get disconnected so make sure you use this function frequently enough.

Here is how it is done:

```
icq_KeepAlive(&link);
```

4.5 Listen on the network with `icq_Main`

As we mentioned before, `icq_Main` (line 227) is very important in that aspect that it listens on the network and activates the correct callbacks.

```
icq_Main(&link);
```

This function is *non blocking*. It does, however, call the callback functions. So if you want a block free implementation, make sure your callbacks are non-blocking as well.

We estimate that for "smooth" operation you would like to call `icq_Main` every few seconds.

4.6 Manage the connection using the "active" functions

Now that your connection is up and running, you can just use the "active" functions all you want. The interface of these functions in very simple and in most cases you can easily figure out what the inputs should be.

4.7 Closing the connection

After we have had our fun with the ICQ connection, we want to terminate the connection. To do this we need to logout from the network and disconnect from the server. The first is done using `icq_Logout` (line 221) and the second is done using `icq_Disconnect` (line 230).

We simply do:

```
 icq_Logout(&link);
icq_Disconnect(&link);
```

And we're out.

We still need to clean our `ICQLINK` in order to free allocated resources.

4.8 Cleaning up with `icq_Done`

This is a technical but very important part. You just clean your `ICQLINK` using `icq_Done` (line 219):

```
icq_Done(&link);
```

If you used dynamic allocation to allocate your `ICQLINK` don't forget to clean that as well.

5 Managing the contact list

The whole point of having an ICQ client is being able to see when your friends are online. For that we need to manage a contact list.

The ICQ network leaves the management of contact list to the client program. Meaning you don't *really* need someone's authorization in order to add them to your contact list.

Supporting authorization requests and replies can easily be done using the appropriate callbacks and functions. In this document, however, we will explain how the contact lists themselves are maintained and managed in icqlib.

Three "active" functions (lines 231–233):

5.1 The list of contacts

The actual data structure that holds the contact list is maintained using the four functions in lines 274–277:

```
 void icq_ContactAdd(ICQLINK *link, unsigned long cuin);
void icq_ContactRemove(ICQLINK *link, unsigned long cuin);
void icq_ContactClear(ICQLINK *link );
void icq_ContactSetVis(ICQLINK *link, unsigned long cuin,
unsigned char vu);
```

Use `icq_ContactAdd` to add a UIN to the list and `icq_ContactRemove` to remove a UIN from the list.

We did not use the other two functions in the csicq project but it is our guess that `icq_ContactClear` will clear the entire contact list and `icq_ContactSetVis` will toggle your visibility to the contact. We do not know, however, what the argument `unsigned char vu` stands for.

5.2 Sending contacts to the network

In order for the ICQ network to tell us the status of users in our contact list, we need to notify the network what the contact list is.

To do that, we have three "active" functions which can be found in lines 231–233:

```
 void icq_SendContactList(ICQLINK *link);
void icq_SendVisibleList(ICQLINK *link);
void icq_SendNewUser(ICQLINK * link, unsigned long uin);
```

When you have all your contacts in your contact list, just let the server know of that by calling `icq_SendContactList`. This should be done after you have logged in to the ICQ network.

In the csicq project, we did not use `icq_SendVisibleList` but we suspect this should be done after you have changed the visibility settings for a contact.

When you add a new user to the contact list, you need to let the server know that this user has been added to the contact list (so the server sends you their status). To do this just call `icq_SendNewUser` with the user's UIN and the ICQ network will be notified.

5.3 Notifications from the ICQ network

We get messages from the ICQ network using three callback functions which can be found in lines 150–153:

```
 void (*icq_UserOnline)(struct icq_link *link, unsigned
long uin, unsigned long status, unsigned long ip, unsigned
short port, unsigned long real_ip, unsigned char tcp_flag
);
void (*icq_UserOffline)(struct icq_link *link, unsigned
long uin);
void (*icq_UserStatusUpdate)(struct icq_link *link, unsigned
long uin, unsigned long status);
```

But what does the network notify *about*?

When a user in your contact list goes online, the `icq_UserOnline` function is called with the user's UIN and connection details (so you know to "paint him in blue" for example).

When a user in the contact list goes offline, the `icq_UserOffline` function is called with the UIN of the user (so you know to "paint him in red").

When the status of a user in the contact list changes, the `icq_UserStatusUpdate` callback is called with the UIN of the user and its new status. From our experience, if you assign a callback function to this handler, the other two functions (`icq_UserOnline` and `icq_UserOffline`) will not be called.