4-12-2006

Structural Analysis of the Log Files of the ICQ Client Version 2003b

Kim Morfitt
Edith Cowan University

Recommended Citation

Morfitt, K. (2006). Structural Analysis of the Log Files of the ICQ Client Version 2003b. DOI: https://doi.org/10.4225/75/57b13687c7055

DOI: 10.4225/75/57b13687c7055
4th Australian Digital Forensics Conference, Edith Cowan University, Perth Western Australia, December 4th 2006.

Structural Analysis of the Log Files of the ICQ Client Version 2003b

Kim Morfitt
Edith Cowan University
Western Australia
kmorfitt@student.ecu.edu.au

Abstract

Instant messenger programs can generate log files of user interactions which are of interest to forensic investigators. Some of the log files are in formats that are difficult for investigators to extract useful and accurate information from. The official ICQ client is one such program. Users log files are stored in a binary format that is difficult to understand and often changes with different client versions. Previous research has been performed that documents the format of the log files, however this research only covers earlier versions of the client. This paper explores the 2003b version of the ICQ client. It documents the analysis process that was undertaken, the files found, much of their structure, and the structure of the records found within. It attempts to provide an accurate and reasonable description of any issues and presents possible solutions to those issues. Finally a brief conclusion is provided which lists outstanding issues.

Keywords

ICQ, instant messaging, log file, forensic, extraction

INTRODUCTION

Instant messenger programs and other messaging systems allow users to interact in real time over the internet. These programs can be used by criminals and Internet predators such as paedophiles to commit crimes or aid in the commission of crimes .

Many of these programs, such as ICQ, are capable of logging user interactions with other users. Some messaging systems, such as the IRC client mIRC, log user interactions to plain text files. Text format log files from chat programs are technically straight forward for a forensic analyst to extract information from. Text editors with search capabilities can be used to find and extract information. Other chat programs encrypt log files, or log to binary or similar. While this may allow for greater log file security, or ease of manipulation of log files by the systems developers, it can greatly increase the complexity and difficulty of extraction and analysis of log files for forensic purposes.

The ICQ instant messaging programs are one such system that log to binary format. For those uninitiated in the various formats of different versions of the ICQ client, it would be difficult and time consuming to extract useful forensic information. The format of the ICQ log files has been documented for a range of versions of the client, from 99a to 2003a by Soeder and Strickz . This information has been applied by Morfitt and Valli to develop techniques for reconstructing log files from fragments and to develop a tool for extracting log file records.

The documentation and techniques described by Soeder, Strickz, Morfitt and Valli cover ICQ client versions 99a to 2003a. There have been subsequent ICQ client versions that are not covered by this documentation. These include among others the ICQ client versions 2003b, 4, and 5. The log files of version 2003b, and possibly versions 4 and 5, differ radically from previous versions. Current documentation cannot be applied to 2003b. It is unknown if the documentation can be applied to versions 4 and 5.

This paper provides a basic analysis of the structure of log files created using the 2003b ICQ client. It identifies the files that make up the users logs, and documents many of the data structures found inside those files. While not all structures are identified, those structures that are most likely to be of immediate forensic interest are

identified and documented. Data in the form of generated log files were collected by conducting a chat between two 2003b clients and analysed. The results of the analysis can be found below.

There are time and resource limits to this research, therefore only those structures that could be identified and documented in the time available are included. As described below, some files did not have enough information to be analysed in the time available. This paper only provides enough analysis on the purpose of various structures to support the arguments made within the paper. If a field, record, or file cannot be reasonably identified, then it is left unidentified.

DATA COLLECTION

Data collection was carried out using two Windows XP computers on a network connected to the Internet. Preparation for data collection was simple. Both computers first had md5 hashing software installed. Next a batch file was created that would generate md5 hashes for all the files in a directory. Lastly ICQ 2003b was installed on both computers. On installation ICQ allows users to create new user accounts. This was required with only one of the installations as the other computer already had user details for an account used in previous tests.

Once the ICQ installations were completed and the users logged in to their new installations, the ICQ log file directory was located and all the files in that directory hashed using the script previously created. The script saves the hashes to a text file in the directory being hashed. Once the hashes were created, a compressed zip copy of each log directory was taken.

Next each user account was added as a contact of the other account. Once this was performed the files in the log directory were again hashed and zipped copies taken. Next a short conversation was initiated between the clients. After each message was sent or received, the files in the log directory were again hashed and zipped copies taken.

Once a few messages had been sent in either direction, it was though enough information had been collected to give the initial insights into the ICQ log file format. The files were copied to the work machine and then unzipped into a working where they could be examined using a hex editor.

INITIAL ANALYSIS

The unzipped directory contained twelve files, not including the hash file. Each file name had a prefix string followed by the users Universal Internet Number (UIN) number, and then a file extension. The UIN is the user's unique identifier on ICQ. There were four different prefixes and three different extensions. Each prefix was found on three files, each of which had one of the three different prefixes.

The prefixes were "Messages", "O", "Plugin", and "Users". The extensions were "cdx", "dbf", and "fpt". These will be referred to as CDX, DBF, and FPT respectively. While none of the extensions gave much of a clue as to the purpose of the file, the prefixes suggested that message records would be found in one or more of the files with the "Message" prefix, configuration information in files with the Plugin prefix, and user and/or contacts information in the file with the Users extension.

The size of the files was quite small, with only two being more than a handful of kilobytes in size. Figure 1 contains a list of the files and their sizes at four points in the data collection process. The first column of sizes is the initial size of the log files. The second is after the contact is added. The third is after the first message was sent or received and the fourth column contains the final size after the last message was sent or received. What this shows us is that when the first contact was added. The hash after the file size indicates that the md5 hash for that file changed from the hash of the file at the previous column.

File Name	Start	Contact Added	First Message	Last Message
Messages192342943.cdx	9,216	9,216*	9,216*	9,216*
Messages192342943.dbf	616	2,041*	2,516*	4,891*
Messages192342943.fpt	512	2,432*	3,072*	5,504*

O192342943.cdx	4,608	4,608	4,608	4,608
O192342943.dbf	514	514*	514	514
O192342943.fpt	167,680	224,000*	224,000	224,000
Plugin192342943.cdx	6,144	6,144*	6,144	6,144
Plugin192342943.dbf	1,284	1,682*	1,682	1,682
Plugin192342943.fpt	1,280	1,664*	1,664	1,664
Users192342943.cdx	10,752	10,752*	10,752	10,752
Users192342943.dbf	616	1,163*	1,163	1,163
Users192342943.fpt	9,216	1,408*	1,408	1,408

Figure 1. File changes at various stages of the analysis

Note that all the files except O192342943.cdx change when the contact is added, and only the three files with the "Message" prefix change when messages are sent or received.

Once all the files were opened the CDX files all showed information at regular intervals of information surrounded by zeroed bytes, suggesting pre-allocated file space. The DBF and FPT files all appeared to contain record information, with the Messages DBF and FPT files showing what appeared to be message records.

FILE ANALYSIS

As mentioned before there are three files with the "Message" prefix in the file name. Each file has a different extension: DBF, FPT, and CDX. When opened up in a hex editor it was shown that the DBF and FPT files contained message records. Each message record appears to be split over the DBF and FPT files. The purpose of the CDX file was not determined. We start with the Message DBF file as this contains the most easily recognisable information.

DBF Message File

Looking through the file, the message information, time stamps, and UIN of the other party in the conversation were much more easily readable than in previous versions. Also the value 0x20 was found throughout the message records. The file, although written in binary rather than text, contains ASCII values. For example the UIN 46512316 when written to a message record in previous ICQ log file versions would be seen in a hex editor as the hexadecimal values BC B8 C5 02. This is the hexadecimal little endian integer version of the UIN. The 2003b version of the UIN is seen in a hex editor as 34 36 35 31 32 33 31 36 34. This is the ASCII version of the UIN in hexadecimal.

Next the start of the file was looked at. Previous versions of the ICQ log files had header information at the start of the file. The 2003b version of the DBF message file is no different. The first 616 bytes of the DBF file appears to contain some form of message accounting information or header. The header appears to be divided up into 32 byte blocks. The first 32 byte block is different from the other blocks, of which there are 10, making 11 blocks which contained information. The remaining header bytes were zeroed out.

Comparing the headers of two different DBF message files shows only one change in the headers at the 5th byte of the header. This change reflected the number messages in the file. The 9th and 10th bytes gave the length of the file header in little endian short integer format. The 11th and 12th bytes contained the length of the messages as a little endian short integer. The messages are padded to maintain a consistent record length. No further information was obtained from this first block of 32 bytes.

The next 10 blocks of 32 bytes all followed a consistent format. They started with a string at the 1st bye of the block, with a hexadecimal number at the 12th and 13th bytes. The last 7 blocks had a hexadecimal number at byte 14. Byte 17 contained a hexadecimal number. All the other bytes were zeroed out, except for the 19th byte of the block that contained the string "SUBJECT", which had a hexadecimal number.

What this suggests is that each of these started with a string of up to 11 bytes, followed by up to 5 bytes of what appears to be a little endian number. Most likely this is actually a 2 or 4 byte integer, as this would fit the 32 bit architecture of 16 bit short integers, and 32 bit long integers. This also appears to fit with the numbers

themselves. It was noted that each of the numbers at the 13th byte in the 32 blocks was larger then the corresponding number in the previous block, if the number was interpreted as a little endian integer. If the 12th byte was a separate number then the numbers at the 13th and 14th bytes form a little endian integer increasing in value for each block.

The number at the 17th block added to the converted little endian number at the 13th byte equalled the converted two byte little endian number at the 13th byte of the subsequent block. This suggested that the number at the 13th byte may be an offset, and that the number at the 17th byte is the length in bytes of a record field at that offset. The numbers at the 12th byte often repeated in different blocks, with only 5 different numbers. This is possibly flag value.

To determine if these were in fact offsets, what the possible offsets are pointing at must be found. The first and most obvious place to look at is the messages themselves. Each message record is the same length and the largest value in the 10 blocks is 4 less than the length in bytes of a record.

The 10 strings in the blocks are ID, USERID, SUBJECT, FOLDERID, GUID, EXTID, TIME, UNREAD, ALL, and ALL2. Some of these strings can be easily matched to potential fields in a message record. ID could be a message id number. USERID could be the UIN of the person the message was sent to, or received from. TIME, suggests the time the message was sent/received. UNREAD, suggests whether the message was read or not.

Looking at the message records it is easy to see the UIN in the record. The byte before the UIN has the value of 1 in the first message. This byte increases by one in each successive message that was sent or received, and in the order that the message was sent or received. This would correlate to our ID number. The offset would be 0x01. That fits with the message that we are looking at.

The next part of the message is the UIN, our possible USERID. The value for this is 0x15, or 21 bytes. This does not immediately correlate to our message. There are 20 bytes of 0x20 (ASCII space) before our suspected ID number. This does not quite correlate either, as byte 21 is the byte at which our potential message ID number is. A large number of messages can be sent by a user. Using only 1 byte to store message numbers means that byte would only allow for 255 messages to be sent. It does not fit the purpose of the record field for it to be only one byte, suggesting there are 21 bytes for message ID numbers.

The next string to consider is the string SUBJECT. This has no direct correlation. After the UIN number and the first byte of the message text is 119 bytes of 0x20 (ASCII space). The offset into the record of the first byte before the message is the same as the possible offset, the short little endian value starting at the 13th byte of the SUBJECT header block. The string itself is stored in short integers, with two bytes per letter, suggesting Unicode format.

The next string is FOLDERID. The offset from the header block of 0x193 gives us the last byte of the message string, assuming that string is a null terminated UTF string. After FOLDERID is the string GUID, which like FOLDERID the offset from the header block of 0x193 gives the offset before an actual GUID.

The strings EXTID, TIME, UNREAD, ALL, and ALL2 all appear to have little endian integer numbers starting at the 13th byte of their respective header blocks which represents an offset into a message record of a field of that type. Supporting this is another number at the 17th byte of the block which gives the length of that field in the record. The only anomaly is that the ID field does not match with what appears to be the start of the record. It suggests that the first byte of the record is a flag marking the start of a record with the byte 0x20. If this is the case then all of the numbers match correctly.

Analysing Other DBF Files

To test this further and possibly explain the anomaly of the ID string block the other three DBF files were analysed. Each appeared to have the same structure.

The O prefixed DBF file was the first looked at. The first 32 bit block was compared against the corresponding Message DBF. The three fields identified as number of records, file header size and record length showed

differences between the two files. The O file had the little endian integer values of 0x02, 0x0188, and 0x3D respectively in each of these fields. These values matched number of records in the O file, the O file header size and record length for records in the O file respectively.

Next the following 32 bit header blocks were looked at. There were three blocks found: ID, GUID, and ALL. The rest of the header was zeroed out. The values at the 13th byte of each block were 0x01, 0x15, and 0x39 respectively. The values at the 17th byte of each block were 0x14, 0x24, and 0x04 respectively. The values at the 13th byte of the GUID and ALL blocks are the sum of the values of the 13th and 17th byte of the previous record. This is the same as the Messages DBF file.

When these numbers were compared against the records, the same characteristics were found as in the Messages DBF file. If the first byte of the record is taken to be a flag value, the values at the 13th byte of the header blocks, excluding the first 32 byte block of the file, represent offsets from that flag for a field within the record. The contents of the field are described by the string at the start of its associated header block.

The next file was the Plugin DBF file. Again looking at the first 32 bytes in a hex editor, the three fields identified as number of records, file header size and record length were all different, with little endian integer values of 0x06, 0x01E8, and 0xC7 respectively. This matched with number of records, header size and record size. Six 32 byte blocks were found after the first block, all of the same format as before. The rest of the header was zeroed out. The strings in the blocks were ID, USERID, GUID, EXTID, DATAID, and ALL. When the offsets at the 13th byte were compared against the records, they match up the same as with the previous DBF files. There is still the off by one with the ID record, suggesting that the first byte may be a start of record marker.

The last file was the Users DBF. The 32 byte block in the file showed the same format as the other DBF files. It gave the number of record entries as 1, the size of the header block as 0x268, and the size of the records as 0x232. All of these values were correct. The three values matched the ID, USERID, ALIAS, FIRSTNAME, LASTNAME, MAINLIST, VISIBLE, INVISIBLE, IGNORE, and ALL. When the offsets at the 13th byte were compared against the records, they match up the same as with the previous DBF files. There is still the off by one with the ID record, suggesting that the first byte may be a start of record marker.

What this shows is that the four DBF files Messages, O, Plugin and Users all follow the same format. Users stores contact information, Messages stores message information, Plugin probably stores plugin information; however the contents of O was not identified. More information could be gained about the purpose of these files by checking the Windows registry for the GUID string entries. This may yield some clues.

Other information is that the DBF files have space that is allocated on demand, and that each record is immediately followed by another record. Fields within a record that do not completely fill that field may be padded with 0x20 (ASCII space).

```
================================================================
== Format of DBF File header (Header Length Varies):
================================================================
00000000        LONG            Unknown, but always 0x30 0x06 0x09 0x1A
00000004        LONG            Number of Records
00000008        WORD            File Header Length
0000000A        WORD            Record Length
0000000C        20 BYTES        Unknown, but always 0x03 0x00 0x00 0x02
                                0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
                                0x00 0x00 0x00 0x00 0x03 0x00 0x00 0x00
```

Figure 2. DBF file Header format

== Format of DBF File Header Field Block (32 BYTES):		
00000000	ASCII	Field Name (bytes after name zeroed out)
0000000B	BYTE	Unknown
0000000C	LONG	Offset into record
00000010	WORD	Field Length
0000000C	14 BYTES	Unknown, but mostly zeroed out

Figure 3. DBF file header field block format

FPT Message Files

The Message FPT was opened in a hex editor. The first thing that was noticed was that it contained message information including message text, similar to the DBF file. What was immediately apparent about the records in the file was that the numbers in the records were probably in big endian format. This was due to the way numbers were shown at the start of a record. The hexadecimal values 0x00, 0x00, 0x00, and 0x01 were present at the start of each record. This is consistent with a 32 bit integer value of 1 being stored in big endian format.

The following four byes showed another probable 32 bit hexadecimal number. Different records showed that the longer the record, the larger the number, suggesting a field for record length. This was confirmed by selecting messages and then calculating the expected end of the message and comparing it with the actual end of the message. What this showed was that the field gave the number of bytes in the message after the last byte of the length field. This is consistent with previous versions of ICQ user logs, where the first field of a record gave the length of that record not including the length field itself. In this case however, the messages appeared to have a field before the message length field that is also not included in the length of the message. The purpose of this field was not identified.

There were what appeared to be16 records in the Messages FPT, but only 8 messages had been sent. The records appeared to be grouped as pairs. Each record pair contained a part of the message or a copy of it. System messages sent when requesting authorisation to add a user to a contact list contained the message from the user requesting authorisation in the first record, and the message sent from the system to the user receiving the authorisation request message in the second record. Non system messages sent from user to user contained what appeared to be a rich text copy of the message in the first record and a Unicode version in the second message.

Following the message length field in the first record of the pair is 22 bytes that have yet to be identified. The first four bytes were unique to each record. The following 18 were the same for each record. This is roughly consistent with previous versions of the ICQ user logs, which had a 5 field record header, if the last 18 bytes are taken to be a 16 byte field, followed by another 2 byte field. One of the fields in the previous header versions included a record id number. This may help to identify any indexing that may be performed on database files.

Following this is data which closely follows the format of a property block, as described by . There is a little endian integer at the start of the block that contains the number of properties. Following this is a short integer which contains the length of the property name, which is stored as a null terminated ASCII string (ASCIIZ) after the length. The length includes the null terminator. Following this is a single byte which describes the type of the property value.

== Format of Property Block (6 BYTES):		
00000000	WORD	Separator value
00000004	LONG	Number of user properties

Figure 4. Property Block format

```
================================================================
== Format of Property (Variable Length):
================================================================
   00000000      WORD          Length of property name
   00000002      ASCIIZ        Property name
$+00000000       BYTE          Property value type :
                                     100 / 64h  = CHAR
                                     101 / 65h  = BYTE
                                     102 / 66h  = WORD
                                     103 / 67h  = INTEGER
                                     104 / 68h  = DWORD
                                     105 / 69h  = LONG
                                     107 / 6bh  = ASCIIZ (1 WORD + String)
                                     109 / 6dh  = Sub list (see below)
                                     111 / 6fh  = DWORD (length) + BYTE array
 +00000010       <var>         Property value
```

Figure 5. Property format

An example of a property stored in this manner is the time stamp. The first value is the length of an ASCIIZ string. For this property the name is “Time”. Including null terminator, the length is 5. The value is a long integer, for which the type is 0x69. Therefore the values of this property would be:

Name length: 0x05

Name: Time

Type: 0x69

Value: 0x40 0x B6 0x 18 0x45 (little endian long value).

The type values appear to match the type values used in previous version of the ICQ log files, with one exception. The type for the value of the “Body” property has the value 0x70, which as not described by Strickz or Soeder . This appears to contain binary data. What is interesting about this data is that even though most of the bytes are different for different records they contain the same values for the last 4 bytes: 0xFF 0xFF 0xFF 0x00.

The second message in the pair appears to simply contain the long integer value 0x01, followed by the little endian long integer value of the remaining bytes in the record, then text from the message in Unicode format.

There was very little information in the header portion of this file that could be easily interpreted.

ANALYSING OTHER FPT FILES

The next file analysed was the Users FPT file. This file had two records. Each record started with the big endian 0x01 value and a length field. After this field was a hexadecimal value 0xE5. Many of the values between 0xE0 to 0xEF have been used as the first byte of the 16 byte signature found in the record headers of records in previous versions. Checking this byte and the following 15 bytes against known signature values, the signature for a contact record was found.

Following the signature are two bytes which could be a separator value as in the record headers of previous log file versions. The separator value for the last version that logs have been generated for in previous research show that separator value to have the byte values 0x23 0x02, or 547 in decimal. From previous experience with separator values, this suggests that the version of ICQ that generated these logs was version 5.47. The separator value in the set of logs currently being analysed is 0x2C 0x02, or 556, giving a possible version of 5.56. This strongly suggests that this is in fact a separator value.

Following our separator value the record is consistent with other previous contact records in that it has a long integer value of RESU, or “USER” in reverse, followed by another long. In previous versions this long represents

the entry status, and has 6 possible values. In the record being looked at, the value is 0x02, which matches the previous versions value for an “active” or valid record .

Following this value is another long integer value which in previous versions is described by Strickz as being the “*GroupID of contact group containing user*”. What follows after this has been worked out to be a series of property blocks, as described by Strickz, with each block being preceded by a separator value. Each block begins a long value containing the number of properties in the block, with the properties following that value. If the block is empty, the long value is simply 0 with no following property information.

There was very little information in the header portion of this file that could be easily interpreted.

The Plugin FPT file was the next to be looked at. This contained 6 records. These records followed a different format to the two previous FPT files. While the records began with the 32 bit integer value 0x01, the 32 bit length value could not be verified. There was not clear record end at any of the records to verify that this value was the record length. Following this was another 32 bit integer as per the previous two files, a 16 bit signature. Although the first byte of the signature, 0xEC, has not been identified as belonging to a particular record type, the last 15 bytes of the signature contain the bytes used in signatures in previous versions. Following this is the separator value and then a 32 bit little endian integer with the length of the users UIN, followed by the users UIN written out in ASCII values.

The bytes following the UIN could not be determined to follow a consistent format, except that some records appeared to have as their last data elements strings written out in ASCII. These strings were preceded by the length of the string in little endian integer format and had no null terminator.

The final FPT file is the O prefixed file. This file contains two records. Both records start with a header consisting of a starting 0x01 integer, a record length integer, the unknown integer, a signature and a two byte separator value. This is the same header format as found in the previous files. The first byte of the signature value for both records was 0xE4. In previous versions this was found to be for “My Details” record . Following the signature, this record followed a format similar to the Users FPT file, although fields, such as the two fields following the “RESU” string may have different meanings for their values.

Again there was very little information in the header portion of this file that could be easily interpreted. No format diagrams were produced for this file.

```
============================================================
== Format of FPT Message Record Part 1 (Header Length Varies):
============================================================
00000000        LONG            Unknown, but always 0x00 0x00 0x00 0x01
00000004        LONG            Record Length
00000008        LONG            Unknown, possible Record ID
0000000C        16 BYTES        Record Signature
0000001B        WORD             Separator value
0000001D        <var>           Property Blocks
```

Figure 5. FPT Message Record Part 1

```
============================================================
== Format of FPT Message Record Part 2 (Header Length Varies):
============================================================
00000000        LONG            Unknown, but always 0x00 0x00 0x00 0x01
00000004        LONG            Record Length
00000008        <var>           Unicode String
```

Figure 6. FPT Message Record Part 2

```
============================================================
== Format of FPT Users & O Records (Header Length Varies):
============================================================
00000000        LONG            Unknown, but always 0x00 0x00 0x00 0x01
00000004        LONG            Record Length
00000008        LONG            Unknown, possible Record ID
0000000C        16 BYTES        Record Signature
0000001C        WORD            Separator value
0000001E        LONG            Label  = 0x55 0x53 0x45 0x52 'USER' in ASCII
00000022        LONG            Unknown
00000026        LONG            Unknown
0000002A        <var>           Property Blocks
```

Figure 7. FPT Users & O Records

CDX FILES

These files were quite sparse with mostly 0x00 bye values. There was not enough information in these files to make a useful analysis, so they were not analysed. Further research using logs with more records and more intensive analysis will be required to extract the purpose of these files and determine enough of their structure for documentation.

CONCLUSION

It was possible to document much of the structure of the log files, even though there was not much emphasis on understanding the meaning of various records and fields. It can be shown that there are three different basic file formats, DBF, FPT, and CDX files. These three files are grouped together by name to form a database of record information for a particular sub set of log file record types that makes up a user's ICQ logs.

The DBF and FPT files did not present a serious challenge to perform a basic analysis on and provide enough information that further research can expand upon. More research will need to be performed to determine the purpose and format of the CDX files, but it could be suggested from the "dx" portion of the file extension that these files may form a type of index.

FURTHER RESEARCH

More detailed analysis needs to be performed on these results. The information is preliminary, and several details, besides the format of the CDX files, need attention. The issue of the off by one error for in the record offsets in the header blocks of the DBF files needs addressing and possibility of record id numbers records needs to be verified. Records that have binary data that could not be identified, such as the unknown data in the Plugin FPT file, also needs further analysis.

REFERENCES

Morfitt, K. (2005). Enhancing the Forensic ICQ Logfile Extraction Tool.

Morfitt, K., & Valli, C. (2005). AFTER CONVERSATION - AN FORENSIC ICQ LOGFILE EXTRACTION TOOL. Paper presented at the 3rd Australian Computer, Network & Information Forensics Conference, School of Computer and Information Science, Edith Cowan University, Perth, Western Australia.

Soeder, D. (2000). ICQNEWDB. Retrieved Jan 20th, 2004

Strickz. (2002). ICQ Db Specs. Retrieved Jan 21st, 2004, from http://cvs.sourceforge.net/viewcvs.py/*checkout*/miranda-icq/Plugins/import/docs/import-ICQ_Db_Specs.txt?content-type=text%2Fplain&rev=1.9

COPYRIGHT