

```
# Copyright (c) 1998-9 America Online, Inc. All Rights Reserved.
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation; either version 2
# of the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

Version: TOC1.0

This document describes the protocol between TOC and TOC clients. The protocol is built on TCP. Framing is done by SFLAP, described at the bottom of this document. Inside each SFLAP frame is a TOC command.

The TOC protocol is ASCII based, and special attention must be placed argument separation. The separator and the rules of separation are different for messages inbound to TOC and outbound to the client. The rules of separation are described in sections below.

The TOC server is built mainly to service the TIC and TiK clients. Since the TIC client is a Java applet, and downloadable, TOC will NOT support multiple TOC protocol versions at the same time. Therefore, TiK users will be forced to upgrade if the protocol version changes. TOC sends down the protocol version it expects the client to speak and understand. Note, the protocol version is a string.

#### Important Notes

=====

- \* TOC will drop the connection if a command exceeds the maximum length, which is currently 2048 bytes. So the client needs to spend special attention to im, chat, and config message lengths. There is an 8k length maximum from TOC to the client.
- \* No commands should be sent to TOC (besides toc\_signon) before a SIGN\_ON is received. If you do send a command before SIGN\_ON the command will be ignored, and in some case the connection will be dropped.
- \* Initial permit/deny items should be sent after receiving SIGN\_ON but before sending toc\_init\_done, otherwise the user will flash on peoples buddylist who the user has denied. You will probably want to send the toc\_add\_buddies at this time also.
- \* After TOC sends the PAUSE message to a client, all messages sent to TOC will be ignored, and in some cases the connection will be dropped. Another SIGN\_ON message will be sent to the client when it is online again. The buddy list and permit/deny items must be sent again, followed by the toc\_init\_done. In most cases the SIGN\_ON message will be sent between 1-2 seconds after the PAUSE message. Therefore a client could choose to ignore the PAUSE message and hope nothing bad happens.

Client -> TOC

=====

The commands and the arguments are usually separated by whitespaces. Arguments with whitespace characters should be enclosed in quotes. Dollar signs, curly brackets, square brackets, parentheses, quotes, and backslashes must all be backslashed whether in quotes or not. It is usually a good idea just to use quotes no matter what. All user names from clients to TOC should be normalized (spaces removed and lowercased), and therefore are the one exception to the always use quotes rule.

When sending commands to the server you will not get a response back confirming that the command format was correct or not! However in some cases if the command format was incorrect the connection will be dropped.

RoastingString="Tic/Toc"

toc\_signon <authorizer host> <authorizer port> <User Name> <Password>  
<language> <version>

The password needs to be roasted with the Roasting String if coming over a FLAP connection, CP connections don't use roasted passwords. The language specified will be used when generating web pages, such as the get info pages. Currently the only supported language is "english". If the language sent isn't found, the default "english" language will be used. The version string will be used for the client identity, and must be less than 50 characters.

Passwords are roasted when sent to the host. This is done so they aren't sent in "clear text" over the wire, although they are still trivial to decode. Roasting is performed by first xoring each byte in the password with the equivalent modulo byte in the roasting string. The result is then converted to ascii hex, and prepended with "0x". So for example the password "password" roasts to "0x2408105c23001130"

toc\_init\_done

Tells TOC that we are ready to go online. TOC clients should first send TOC the buddy list and any permit/deny lists. However toc\_init\_done must be called within 30 seconds after toc\_signon, or the connection will be dropped. Remember, it can't be called until after the SIGN\_ON message is received. Calling this before or multiple times after a SIGN\_ON will cause the connection to be dropped.

toc\_send\_im <Destination User> <Message> [auto]

Send a message to a remote user. Remember to quote and encode the message. If the optional string "auto" is the last argument, then the auto response flag will be turned on for the im.

toc\_add\_buddy <Buddy User 1> [<Buddy User2> [<Buddy User 3> [...]]]

Add buddies to your buddy list. This does not change your saved config.

toc\_remove\_buddy <Buddy User 1> [<Buddy User2> [<Buddy User 3> [...]]]

Remove buddies from your buddy list. This does not change your saved config.

toc\_set\_config <Config Info>

Set the config information for this user. The config information is line oriented with the first character being the item type, followed by a space, with the rest of the line being the item value. Only letters, numbers, and spaces should be used. Remember you will have to enclose the entire config in quotes.

Item Types:

- g - Buddy Group (All Buddies until the next g or the end of config are in this group.)
- b - A Buddy
- p - Person on permit list
- d - Person on deny list
- m - Permit/Deny Mode. Possible values are
  - 1 - Permit All
  - 2 - Deny All
  - 3 - Permit Some
  - 4 - Deny Some

toc\_evil <User> <norm|anon>

Evil/Warn someone else. The 2nd argument is either the string "norm" for a normal warning, or "anon" for an anonymous warning. You can only evil people who have recently sent you ims. The higher someones evil level, the slower they can send message.

toc\_add\_permit [ <User 1> [<User 2> [...]]]

ADD the following people to your permit mode. If you are in deny mode it will switch you to permit mode first. With no arguments and in deny mode this will switch you to permit none. If already in permit mode, no arguments does nothing and your permit list remains the same.

toc\_add\_deny [ <User 1> [<User 2> [...]]]

ADD the following people to your deny mode. If you are in permit mode it will switch you to deny mode first. With no arguments and in permit mode, this will switch you to deny none. If already in deny mode, no arguments does nothing and your deny list remains unchanged.

toc\_chat\_join <Exchange> <Chat Room Name>

Join a chat room in the given exchange. Exchange is an integer that represents a group of chat rooms. Different exchanges have different properties. For example some exchanges might have room replication (ie a room never fills up, there are just multiple instances.) and some exchanges might have navigational information, and some exchanges might have ... Currently exchange should always be 4, however this may change in the future. You will either receive an ERROR if the room couldn't be joined or a CHAT\_JOIN message. The Chat Room Name is case insensitive and consecutive spaces are removed.

toc\_chat\_send <Chat Room ID> <Message>

Send a message in a chat room using the chat room id from CHAT\_JOIN. Since reflection is always on in TOC, you do not need to add the message to your chat UI, since you will get a CHAT\_IN with the message. Remember to quote and encode the message.

toc\_chat\_whisper <Chat Room ID> <dst\_user> <Message>

Send a message in a chat room using the chat room id from CHAT\_JOIN. This message is directed at only one person. (Currently you DO need to add this to your UI.) Remember to quote and encode the message. Chat whispering is different from IMs since it is linked to a chat room, and should usually be displayed in the chat room UI.

toc\_chat\_evil <Chat Room ID> <User> <norm|anon>  
Evil/Warn someone else inside a chat room. The 3rd argument is either the string "norm" for a normal warning, or "anon" for an anonymous warning. Currently chat evil is not turned on in the chat complex.

toc\_chat\_invite <Chat Room ID> <Invite Msg> <buddy1> [<buddy2> [<buddy3> [...]]]  
Once you are inside a chat room you can invite other people into that room. Remember to quote and encode the invite message.

toc\_chat\_leave <Chat Room ID>  
Leave the chat room.

toc\_chat\_accept <Chat Room ID>  
Accept a CHAT\_INVITE message from TOC. The server will send a CHAT\_JOIN in response.

toc\_get\_info <username>  
Gets a user's info a GOTO\_URL or ERROR message will be sent back to the client.

toc\_set\_info <info information>  
Set the LOCATE user information. This is basic HTML. Remember to encode the info.

toc\_set\_away [<away message>]  
if the away message is present, then the unavailable status flag is set for the user. If the away message is not present, then the unavailable status flag is unset. The away message is basic HTML, remember to encode the information.

toc\_get\_dir <username>  
Gets a user's dir info a GOTO\_URL or ERROR message will be sent back to the client.

toc\_set\_dir <info information>  
Set the DIR user information. This is a colon separated fields as in:  
"first name":"middle name":"last name":"maiden name":"city":"state":"country":"email":  
"allow web searches"  
Should return a DIR\_STATUS msg. Having anything in the "allow web searches" field allows people to use web-searches to find your directory info. Otherwise, they'd have to use the client.

toc\_dir\_search <info information>  
Perform a search of the Oscar Directory, using colon separated fields as in:  
"first name":"middle name":"last name":"maiden name":"city":"state":"country":"email"  
Returns either a GOTO\_URL or ERROR msg.

toc\_set\_idle <idle secs>  
Set idle information. If <idle secs> is 0 then the user isn't idle at all. If <idle secs> is greater than 0 then the user has already been idle for <idle secs> number of seconds. The server will automatically keep incrementing this number, so do not repeatedly call with new idle times.

toc\_set\_caps [ <Capability 1> [<Capability 2> [...]]]  
Set my capabilities. All capabilities that we support need to be sent at the same time. Capabilities are represented by UUIDs.

toc\_rvous\_propose - Not Implemented Yet

toc\_rvous\_accept <nick> <cookie> <service> <tlvlist>  
Accept a rendezvous proposal from the user <nick>. <cookie> is the cookie from the RVOUS\_PROPOSE

message. <service> is the UUID the proposal was for. <tlvlist> contains a list of tlv tags followed by base64 encoded values.

toc\_rvous\_cancel <nick> <cookie> <service> <tlvlist>  
Cancel a rendezvous proposal from the user <nick>. <cookie> is the cookie from the RVOUS\_PROPOSE message. <service> is the UUID the proposal was for. <tlvlist> contains a list of tlv tags followed by base64 encoded values.

toc\_format\_nickname <new\_format>  
Reformat a user's nickname. An ADMIN\_NICK\_STATUS or ERROR message will be sent back to the client.

toc\_change\_passwd <existing\_passwd new\_passwd>  
Change a user's password. An ADMIN\_PASSWD\_STATUS or ERROR message will be sent back to the client.

TOC -> Client

=====

All user names from TOC to client are NOT normalized, and are sent as they should be displayed. String are NOT encoded, instead we use colons as separators. So that you can have colons inside of messages, everything after the colon before :<Message> should be considered part of the message (ie don't just "split" on colons, instead split with a max number of results.)

SIGN\_ON:<Client Version Supported>

This is sent after a successful toc\_signon command is sent to TOC. If the command was unsuccessful either the FLAP connection will be dropped or you will receive a ERROR message.

CONFIG:<config>

A user's config. Config can be empty in which case the host was not able to retrieve it, or a config didn't exist for the user. See toc\_set\_config above for the format.

NICK:<Nickname>

Tells you your correct nickname (ie how it should be capitalized and spacing)

IM\_IN:<Source User>:<Auto Response T/F?>:<Message>

Receive an IM from some one. Everything after the third colon is the incoming message, including other colons.

UPDATE\_BUDDY:<Buddy User>:<Online? T/F?>:<Evil Amount>:<Signon Time>:<IdleTime>:<UC>

This one command handles arrival/depart/updates. Evil Amount is a percentage, Signon Time is UNIX epoc, idle time is in minutes, UC (User Class) is a two/three character string.

uc[0]:

' ' - Ignore  
'A' - On AOL

uc[1]

' ' - Ignore  
'A' - Oscar Admin  
'U' - Oscar Unconfirmed  
'O' - Oscar Normal

uc[2]

'\0' - Ignore  
' ' - Ignore  
'U' - The user has set their unavailable flag.

ERROR:<Error Code>:Var args

\* General Errors \*

- 901 - \$1 not currently available
- 902 - Warning of \$1 not currently available
- 903 - A message has been dropped, you are exceeding the server speed limit

\* Admin Errors \*

- 911 - Error validating input
- 912 - Invalid account
- 913 - Error encountered while processing request
- 914 - Service unavailable

\* Chat Errors \*

- 950 - Chat in \$1 is unavailable.

\* IM & Info Errors \*

- 960 - You are sending message too fast to \$1
- 961 - You missed an im from \$1 because it was too big.
- 962 - You missed an im from \$1 because it was sent too fast.

\* Dir Errors \*

- 970 - Failure
- 971 - Too many matches
- 972 - Need more qualifiers
- 973 - Dir service temporarily unavailable
- 974 - Email lookup restricted
- 975 - Keyword Ignored
- 976 - No Keywords
- 977 - Language not supported
- 978 - Country not supported
- 979 - Failure unknown \$1

\* Auth errors \*

- 980 - Incorrect nickname or password.
- 981 - The service is temporarily unavailable.
- 982 - Your warning level is currently too high to sign on.
- 983 - You have been connecting and disconnecting too frequently. Wait 10 minutes and try again. If you continue to try, you will need to wait even longer.
- 989 - An unknown signon error has occurred \$1

EVILED:<new evil>:<name of eviler, blank if anonymous>

The user was just eviled.

CHAT\_JOIN:<Chat Room Id>:<Chat Room Name>

We were able to join this chat room. The Chat Room Id is internal to TOC.

CHAT\_IN:<Chat Room Id>:<Source User>:<Whisper? T/F>:<Message>

A chat message was sent in a chat room.

CHAT\_UPDATE\_BUDDY:<Chat Room Id>:<Inside? T/F>:<User 1>:<User 2>...

This one command handles arrival/departs from a chat room. The very first message of this type for each chat room contains the users already in the room.

CHAT\_INVITE:<Chat Room Name>:<Chat Room Id>:<Invite Sender>:<Message>

We are being invited to a chat room.

CHAT\_LEFT:<Chat Room Id>

Tells tic connection to chat room has been dropped

GOTO\_URL:<Window Name>:<Url>  
 Goto a URL. Window Name is the suggested internal name of the window to use. (Java supports this.)

DIR\_STATUS:<Return Code>:<Optional args>  
 <Return Code> is always 0 for success status.

ADMIN\_NICK\_STATUS:<Return Code>:<Optional args>  
 <Return Code> is always 0 for success status.

ADMIN\_PASSWD\_STATUS:<Return Code>:<Optional args>  
 <Return Code> is always 0 for success status.

PAUSE  
 Tells TIC to pause so we can do migration

RVOUS\_PROPOSE:<user>:<uuid>:<cookie>:<seq>:<rip>:<pip>:<vip>:<port>  
 [:tlv tag1:tlv value1[:tlv tag2:tlv value2[:...]]]  
 Another user has proposed that we rendezvous with them to perform the service specified by <uuid>. They want us to connect to them, we have their rendezvous ip, their proposer\_ip, and their verified\_ip. The tlv values are base64 encoded.

Typical Signon Process

=====  
 Except for the section marked optional this is an sequential process. Each line MUST occur before the following line.

- \* Client connects to TOC
- \* Client sends "FLAPON\r\n\r\n"
- \* TOC sends Client FLAP SIGNON
- \* Client sends TOC FLAP SIGNON
- \* Client sends TOC "toc\_signon" message
- \* if login fails TOC drops client's connection
- else TOC sends client SIGN\_ON reply
- \* if Client doesn't support version it drops the connection

[BEGIN OPTIONAL]  
 \* TOC sends Client CONFIG  
 \* Client sends TOC permit/deny stuff  
 \* Client sends TOC toc\_add\_buddy message  
 [END OPTIONAL]

\* Client sends TOC toc\_init\_done message

SFLAP Documentation

=====  
 SFLAP is pretty much a FLAP connection except the DATA frame payload is a null terminated string when traveling from client to host, it is NOT null terminated when traveling from host to client. The FLAP Header is binary data, and is in network byte order. The data portion is at offset 6, after the header. The sequence number is sequential in each direction. So packets from the server to client have one sequence number, while the packets from the client to server have an independent increasing number.

FLAP Header (6 bytes)

-----

Offset	Size	Type
0	1	ASTERISK (literal ASCII '*')
1	1	Frame Type

2	2	Sequence Number
4	2	Data Length

Valid Frame Type Values

```

-----
1  SIGNON
2  DATA
3  ERROR      (Not used by TOC)
4  SIGNOFF    (Not used by TOC)
5  KEEP_ALIVE

```

TOC SIGNON FRAME TYPE

```

-----
Sequence Number contains the initial sequence number used in each direction.
Data Length contains the payload length, with the payload described
below. The payload area is NOT null terminated.

```

```

Host To Client:
  4 byte FLAP version (1)

```

```

Client To Host:
  4 byte FLAP version (1)
  2 byte TLV Tag (1)
  2 byte Normalized User Name Length
  N byte Normalized User Name (NOT null terminated)

```

TOC DATA FRAME TYPE

```

-----
Sequence Number contains the next sequence number.
Data Length is the length of the payload, including the null termination
from client to host.

```