*** CONFIDENTIAL ***

Updated: 11/18/07 for version 3.0.1.

The following information details what is involved in setting up and running the Pro version of the Ventrilo 3.0 server.

It does not discuss the normal things that are common between the Public and Pro versions. Please refer to the ventrilo_srv.htm file that comes with the Public servers for more general setup information.

################ RegKey ################

The Pro server now requires that a registration key file be provided via a command line option. The registration key in this file will be sent to the Ventrilo authentication server in order to verify the legitimacy of the server and so that we can bill you accordingly.

Each licensed hosting company will receive a single registration key that is unique to them. The format of the file is as follows and is a single line.

```
REGKEY=XX-XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX
```

You need to replace the X's with your registration key. The name and location of the file is entirely up to you.

################ Command line options ################

The following command line options are additional to any of the existing ones from the previous servers i.e. if you are using the –f and –d options on the older servers then you will need to continue using them, but in addition you must include the –r and –i.

Note: Command line switches are not case sensitive. A –r is the same thing as a –R.

You will also need to supply a functional INI file since this package does not include one. These options are additions to what the public and older servers needed to run.

-R

The Ventrilo Pro server must be started with a –Rfilename on the command line in order to locate and read the registration key file. You can place the file anywhere you want in an effort to secure it. As such, when specifying the filename you should provide a fully qualified path and filename. Don't assume that the file will be in the current working directory unless you have taken steps to guarantee this, and it is preferred that the file be in some other location then any of the configuration files.

Note: There is no space between the –R and the specified file name.

-X

If specified it will scramble the contents of the specified registration key file (see the –R option) and then delete the file completely after it has been read and processed. This way you can create a temporary file using a random name and pass the filename on the command line when starting the server, then have the server purge the file. This is another attempt at protecting your registration key information. Please note: This option is not available when using the NT Service program for starting the servers.

-I

Each Pro server must be attached to a specific network interface (IP address). This can be done in one of two ways. 1) Assign an “Intf=” value in the servers INI file. 2) Add the command line option –Iipaddress. Both of these can be either a litteral IP address or a valid hostname which resolves to a real IP address on the machine for which the server is being started on.

Note: There is no space between the –I and the IP address

-U

Optional. Enables overriding the default timeout value before the server updates the USR, TRGCMD and TRGVOICE files. The default timeout is now 1 minute. With this option you can set it to a minimum of 10 seconds. Example –u10. If no value is specific then 10 seconds is assumed.

################ Phone-home requirements ################

Verison 3.0 requires that both the TCP and UDP ports be open for the servers designated operational port as specified in the INI file. Both inbound to the port and outbound from the port must be enabled.

Each Pro server must be able to phone-home to the Ventrilo authentication servers. This requires that a bi-directional exchange of UDP packets be permitted.

The following example is used on a Linux based server to open the required port for communicating with the authentication servers. It was added to an existing /etc/sysconfig/iptables script that was originally created with the lokkit program. You might need to change it to fit your systems network port filtering and firewall scripts.

```
-A RH-Lokkit-0-50-INPUT -p udp -m udp --sport 6000 -d 0/0 -j ACCEPT
```

The following shows how to open the defined server connection port number. It also demonstrates opening a range of ports from 3000 to 4000 with just the two commands.

```
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 3000:4000 --syn -j ACCEPT
-A RH-Lokkit-0-50-INPUT -p udp -m udp --dport 3000:4000 -j ACCEPT
```

The Ventrilo server will always send to port 6000 of the authentication servers. However, the source port that is opened by the server is dynamic. The response from the authentication server will always be sent back to the dynamic port of the Ventrilo server. This will help to deter any potential packet flooding of the system and potential fraud.

If the servers main port is not accessible via UDP then the client will display the following “MSG: Contacting server” and will never go past it. It’s also possible that the clients firewall is restricting UDP access to your server as well. If at least one client can access the server and another can’t then the problem is on the side of the client.

Note: Some firewalls, other then iptables, will need that you open port 6100 as well. 6100 is the destination port number for the synchronization servers. With iptables opening the UDP for the servers main port is sufficient. If this doesn’t work then open 6100 as well.

All clients must be able to talk to port 6100 as well. While most home routers and firewalls should permit this by default its possible that some firewalls might be more restrictive. This will most likely be the case with corporate and school based networks. If a client displays the “MSG: Synchronizing” then it was able to access your servers UDP port but is now probably unable to access the synchronization servers on port 6100.

If a server is “READY” but has not checked in with an authentication server, each client that connects to the server will be presented with the following message.

“This Professional server has not been authenticated yet. Please try connecting again in a few minutes.”

Once a server is authenticated it will remain that way for an undisclosed grace period. Thus, if a server is authenticated and you have been made aware of any possible problems with the authentication system, it’s best not to restart any of the Ventrilo servers.

Under normal conditions each Ventrilo server should be checking in with the authentication servers on a regular basis. Should this regular check in not occur it could possibly trigger fraud detection mechanisms in our billing/reporting system. FYI.

################ Proxies ################

Proxies are not supported nor are they allowed. This has always been our official position. But starting with version 3.0 there are technical reasons why proxies will not be tolerated.

################ Included files ################

The package that included this file also contained several other files for professional hoster use only.

There should be 3 different ventrilo_srv files. One for the Windows platform, one for Linux and one for FreeBSD. Normally, the Ventrilo server would be called "ventrilo_srv" for unix platforms and "ventrilo_srv.exe" for the Windows platform. However, since all of these came with this file it was necessary to rename them. You can change the name back to ventrilo_srv depending on which platform you are running.

Included EXE's might be renamed XEX in order to bypass email spam filters.

################ What not edit ################

Ventrilo_srv.usr

Prior versions of the Ventrilo server would re-read the USR file each time a client connected. This had a side benefit of allowing the hosting company to edit the USR file externally without restarting the server. This is no longer allowed due to changes in the USR file it self and the way the server functions.

Due to the complexity of the USR file in version 3.0 it is no longer practical to re-read the file each time a client connects. Thus the server will read it once when started and it will cache all of the information. Updated information is written back to the file periodically when a user makes changes to user accounts via the new User Editor built-in to the client program it self. For this reason you can never be sure that the USR file contains the most accurate information available. The reason for the deferred updates is because the servers requirement to support 4000 unique accounts and the amount of time to write that much information back to disk can take a while depending on system performance.

In order to edit the USR file you must be sure that the server has been terminated completely and that it was shutdown using a formal technique which signals the server to stop and to update any cached data. With that said, it is strongly recommended that you not edit the USR file at all with the exception of modifying an existing accounts password in case the user were to lock them selves out.

You should never (ever) add accounts, delete accounts or modify anything other then the password of an existing account. If you do you risk damaging the security and access rights of one or more accounts that the user has defined via the client User Editor and other server level configuration tools that the 3.0 clients have built-in.

Note: Should you decide to continue the option of overriding an existing accounts password you can insert a "Password=" key that takes a clear-text version of the password instead of trying to hash the password into it's "EncPass=" form. When the server starts it looks for the Password key first. If found it will hash it into it's encrypted form and start the cache update timer so that the next time the server needs to write to the USR file it will write the encrypted form instead of the clear-text form.

################ What can I edit ################

Rather then listing all of the files that can’t be edited, here is a list of files that you can edit.

Ventrilo_srv.ini

Basic configuration details that you would need to setup.

Ventrilo_srv.ban

In case a user locks them selves out of the server or a bad admin.

Every other file generated by the server should be left alone for several reasons.

1) The file might be cached by the server and would require a formal server shutdown before you could be assured that its contents are up to date.

2) The file has cross linked complexity with the contents of other files and without the knowledge of what the complexity is you risk breaking or damaging the access rights as dictated by the server administrator or one of his lieutenants.

3) Version 3.0 Client programs have very powerful and user friendly interfaces for manipulating server and account configurations. These changes are live and dynamic and do not require a server restart or any intervention of the hosting company. This helps to keep your support time low.

################ Server Admin password ################

It is strongly recommended that you encourage all of your clients to change their global server admin password (the one in the INI file) and restart the server. As they should be doing this through your control panel I would encourage you to enforce a minimum of 8 characters for the password and that it would be a mixture of upper and lower case letters with at least 2 numbers. I would also suggest the use of an even longer password.

Note: If you modify the admin password to be empty or the key not exist then this will in effect make the admin password unavailable and could never be guessed. But the server will still run. This could be a good way to encourage the owner to login to your Control Panel and update the password.

The server now has better logging of people trying to login with someone else’s account (password guessing) and anyone who attempts to use the server admin password. Rather then hunting down the previous ID of the offending user in order to translate it into an IP that is about to be banned, the IP and user name as displayed on the same log entry. Failed server admin logins also contain the word CRITICAL: which can be used in the loggrep rcon command.

################ Timed bans ################

Version 3.0 introduces the concept of advanced client access rights and more checks and balances to prevent possible exploitation of the account or account password guessing.

If a client attempts to use the global server admin password and does not enter the correct password the server will automatically disconnect that client. The client can log back in immediately and try again. If he gets it wrong again he will be kicked one more time.
A total of 3 tries are permitted before the server will enact a “Timed Ban” for the account. The first time the timed ban is triggered that IP address is blocked for 15 minutes. While the timed ban is active the client will receive a notification that they are on the timed ban list and how much time remains before they are allowed back in.

After the timed ban is lifted the client can repeat his attempts to use the admin password. Again they will get 3 tries, but this time the timed ban will be 30 minutes, then 45 minutes, then 1 hour, then 1:15, then 1:30 and so on up to 3 days max.

Timed bans are not recorded in the BAN file. They are cached locally by the server. Restarting the server will flush them, or a server admin can simply select the account name in the User Editor and click the Update button to flush the timed ban. Only a server admin can do this. A regular “Add User” account can not.

These same concepts apply to individual accounts created on the server which have their own unique password. The difference is that a user account login with the wrong password will get 10 tries before they are automatically added to the timed ban list. The timed ban will be 10 minutes and will increase in 10 minute increments. The next time they fail their 10 tries they will be banned 20 minutes, then 30 minutes and so on until a maximum timed ban for logins of 4 hours.

There are several other areas of the program that will trigger timed bans that vary in their trigger counts, intervals and maximum times.

Each of the timed bans no matter what triggered it can be manually lifted using the Server admin Update trick mentioned above.

################ Channel specific codec support ################

Version 3.0 now supports channel specific codecs. This functionality is off by default and must be enabled by the hosting company by adding the following key to the servers INI file.

```
[Server]
CodecMaxBW=10000
```

The number specifies the maximum number of bytes that a can be consumed per second by any given codec. Instead of limiting specific codecs and formats you are simply limiting the amount of BW (Bandwidth) that the codec is allowed to use per second for any single stream.

The example above of 10000 means that all codecs can be selected as none of them go above 8905 (GSM 44Khz)

You can get the bytes/sec for each codec by starting the server with the -? command line switch.

The only codecs that will be supported by individual channels will be those codecs that have a bytes/sec value less then or equal the CodecMaxBW value.

Performing a formal shutdown

Requesting the server to perform a formal shutdown requires one of the following.

Linux:

Shell command:

```
kill `cat ventrilo_srv.pid`
```

Programmatically:

Send a SIGTERM signal to the process.

Windows .NET:

If you use .NET to spawn the application you can NOT use Process.Kill to shut it down. Process.Kill kills the process without giving it time to flush its cached data. You must use Process.CloseMainWindow or a variation thereof. Here is an example of starting and stopping the process.

```
Process
               *p;
ProcessStartInfo
               *si = new ProcessStartInfo;

printf( "Starting server.\n" );

si->Arguments = S" -rventrilo_srv.reg –ix.x.x.x -fventrilo_srv";
si->FileName = S" ventrilo_srv.exe";
p = Process::Start( si );

printf( "Process ID %d\n", p->get_Id() );
printf( "Sleep for 15 seconds.\n" );

Thread::Sleep( 1000 * 15 );

// Works but not a formal shutdown.
//p->Kill();

// Works but implies a console window exists.
// Would not happen in detached mode.
// Might exist in background if spawned from a service.
p->CloseMainWindow();

if ( p->WaitForExit( 1000 * 20 ) == true )
{
        Console::WriteLine( S"Finished" );
}
else
{
        Console::WriteLine( S"Process did not terminate.” );
        Console::WriteLine( S”Issueing kill." );

        p->Kill();
}
```

Windows API:

Do not use the TerminateProcess API. It suspends all processing and forces the target program to exit immediately.

Here is an example of starting and stopping the process. Note that if the process that spawns the application using the example code exits the system so will each of the sub-processes that it spawned as each of them will automatically receive the break signal by the system. A standard WIN32 implementation using the CloseMainWindow technique above in the .NET example is also possible if you can identify the window and its associated process ID.

```
STARTUPINFO
               si;
PROCESS_INFORMATION
               pi;
char
               cmdline[ _MAX_PATH ],
               args[ _MAX_PATH ];

ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);

ZeroMemory( &pi, sizeof(pi) );

strcpy( cmdline, "ventrilo_srv.exe" );
strcpy( args, " -rventrilo_srv.reg –ix.x.x.x -fventrilo_srv -d" );

// Start the child process.

if ( !CreateProcess(   cmdline,
                       args,
                       NULL,
                       NULL,
                       FALSE,
                       CREATE_NEW_PROCESS_GROUP,
                       NULL,
                       NULL,
                       &si,
                       &pi
                       )
               )
{
       printf( "CreateProcess failed.\n" );
       exit( 1 );
}

printf( "Sleeping for 15 seconds.\n" );
Sleep( 15 * 1000 );

printf( "Sending Ctrl-Break\n" );
GenerateConsoleCtrlEvent( CTRL_BREAK_EVENT, pi.dwProcessId );

// Wait until child process exits.

printf( "Waiting for process to leave.\n" );
WaitForSingleObject( pi.hProcess, INFINITE );

// Close process and thread handles.
CloseHandle( pi.hProcess );
CloseHandle( pi.hThread );
```

You can tell if a formal shutdown has been processed successfully by examining the log file of the server. There should be two lines at the end and both of which have the END: leader. There might be several diagnostics between them showing items that have been flushed.

If you see the “END: Shutdown complete” then you are doing it correctly.