

# **Raven Menu Format v1.00**

## **(The scripted menus used in Soldier of Fortune)**

© Raven Software, 2000

Designed and implemented by John 'Pagan' Scott, BSc. (Hons.), ARCS.

Insert legal crap here > No warranty, implied or otherwise, etc etc, yada, yada.<

I have made every attempt to make sure this document is as error free as possible, however, I cannot guarantee its accuracy. Please email [jscott@ravensoft.com](mailto:jscott@ravensoft.com) if you find any errors, or have any suggestions on how to improve it. I shall endeavour to keep the most recent version online at

“<http://www2.ravensoft.com/users/jscott/sof/rmf.htm>”

## **Overview**

The scripted menus allow for easy manipulation of console variables and configuration files from within a GUI environment. They are interpreted as loaded and create their own layout. No compilation is necessary to change a menu. They sit on top of the console and have no direct access to any data members outside of their own classes. The page is laid out as a bunch of dirty rectangles. The system was designed to be as easy to use as HTML, but extended to take full advantage of the Quake style configuration files and console.

## **Basics**

The first thing you should do is familiarise yourself with the console and its intricacies. The following documentation presumes a thorough knowledge of how the console works and how to get the most from it.

The only requirements for a menu file are –

- (a) The extension is “rmf”
- (b) The file resides in “base/menus” or “user/menus”
- (c) The file contains a “<stm>” to start and a “</stm>” to end.

A description of the internal layout of the code will help define some of the terminology used. There is a controlling class (menu system), which is created when the menus are brought up. This contains all global information required by the menu system and every interface required to fully utilise it. There can be only one menu system class at any one time.

Attached to the menu system class is a linked list of menus. Menus are defined as one complete set of frames. Each menu defines one complete menu screen (e.g. the video settings screen). A menu contains all the information global to that page and grabs anything more from the parent menu system. There can be as many menu pages attached to the menu system as memory will allow.

Each menu page has a linked list of frames attached to it. The frame starts out the full size of the menu page, and frames are cut from it. Frames are named and can be referenced directly. A frame does not necessarily have to have content. There can be as many frames as memory will allow. An example of a frame is the scrollable area in the video settings menu.

Each frame may or may not have a list of pages associated with it. One frame can have many pages attached to it, but the page on the top of the page stack is the only one displayed. There can be an unlimited number of pages attached to any one frame, but, in the vast majority of cases, it is only one.

Each page has a list of areas attached to it. These define the actual content of the whole page (text, models, images etc). Again, there is no limit to the number of areas that can be used as content. Extensive error

checking is performed on each and every area as it is created and loaded. Should anything fail, a developer warning will be issued and the area will simply not appear.

There is a stack of all the above, created once and only destroyed when the menus are exited. So, if a frame or page is revisited, it is not reconstructed, just redisplayed. Pages are only constructed when a page is first visited.

All paths have a presumed base and extension. For example, all images used in the menu system reside in “base/pics/menus/” directory and have the extension “m32”. All ghoul models reside in “base/ghoul/menus/” and have the extension ghl.

If you examine the existing files you will see a great amount of text in the format of “^MENU\_WON\_START^”. Anything within “^” is indirected through a string package for localisation. Unless you wish to translate into many different languages, please ignore this feature.

## Key

Terminology used throughout this document.

**action** – a command that is normally bound to a key and used in game e.g. +attack, itemuse

**align** – either center, left or right.

**bolt** - the name of a bolt (sometimes known as a null) in a ghoul model.

**colour** - a value representing a 32 bit RGBA colour.

**command** – a command that can be issued from the console e.g. disconnect

**cvar** – a console variable name e.g. r\_fullbright

**font** - either small, medium or title.

**frame** - the name of a frame within a screen.

**keyname** - the name used to bind a key. E.g. KP\_DEL, SEMICOLON, A, B or C

**list** - a list of items separated by commas e.g. “Armor,Flash Pak,Grenade”

**menu** - the name of an rmf file

**model** – the name of a ghoul model.

**sp** – the name of a string package.

**text** - a generic text string.

**value** - a number, normally an integer but can be positive or negative and sometimes a fraction. E.g. 2.4

Parameters appearing in <> are required, parameters in [] are optional.

## Console Commands

The menus are activated through a series of commands issued from the console.

### Command: menu <menu> [frame]

This constructs a menu page or redisplay it if it has already been constructed. E.g. “menu main” brings up the main menu. If a frame name is specified, the page will be built in that frame.

### Command: intermission <menu>

Identical to the menu command, with the exception that the created page cannot be exited.

### Command: reloadall

Marks all pages in the stack to be reloaded when they are next visited. Used to change languages.

### **Command: refresh**

Refreshes the topmost page. For example, if a displayed console variable has changed by another process, you will have to refresh the screen to see it in the menu.

### **Command: popmenu**

Pops the most recent page off the top of the stack as long as there is a valid page behind it. If the most recent page was the only page on the stack, then the menu will only be popped if the console is enabled.

### **Command: killmenu**

Identical to popmenu, but always pops the page irrelevant of the validity of the page behind or the availability of the console.

### **Command: return**

Destroys the whole menu system provided there is an active game running in the background. I.e returns to the game.

### **Command: menuoff**

Destroys the whole menu system including all pages currently on the stack.

### **Command: select <cvar>**

Used to select items. Used exclusively with the selection area.

### **Command: checkpass <password> <pass menu> <fail menu>**

Checks **password** against the parental lock password and displays the **pass menu** if they match or the **fail menu** if they don't

### **Command: changepass <oldpass> <newpass> <verify newpass> [pass menu] [fail menu]**

If **oldpass** matches the parental lock password, and **newpass** is the same as **verify newpass**, then the parental lock password will be changed. If **pass menu** is set, then this will be displayed if everything was ok, and if **fail menu** is set, then this will be displayed if any error occurred.

### **Command: freshgame**

Starts a fresh game, bringing up a requester asking if you wish to exit the current game if you are in one.

## **Main Keywords**

The basic keywords are split up into three main categories, control (how the pages mix together, how big they are), layout (how the various areas are laid out) and areas (the actual content of the page). A brief summary of all the main keywords follows.

### **Control: stm**

The main keyword used to define a menu page. Must be paired with a /stm.

### **Control: frame <name> <width> <height>**

Defines a frame for a page.

### **Control: key <keyname> <command>**

Allows actions to be bound conditional to a user pressing a key anywhere on the page.

### **Control: ikey <action> <command>**

Allows actions to be bound to all keys bound to the action defined in the statement.

### **Control: ckey <keyname> <cvar> <false command> <true command>**

Allows actions to be bound conditional to a user pressing a key and the contents of a console variable.

### **Control: comment <text>**

Can be used to insert a comment into a menu

### **Control: includecvar <cvar>**

Include the text contained in a console variable. This can contain control sequences.

### **Control: include <page>**

Always includes another menu at this point in the current page.

### **Control: cinclude <cvar> <page>**

Includes another menu at this point in the current page conditional on a console variable being set.

### **Control: cninclude <cvar> <page>**

Includes another menu at this point in the current page conditional on a console variable being clear.

### **Control: exinclude <cvar> <page> <page>**

Includes one of two menus at this point in the current page dependent on a console variable being set or clear.

### **Control: einclude <cvar> <value> <page>**

Includes another menu at this point in the current page conditional on a console variable matching exactly a value.

### **Control: alias <action> <command> [command]**

Sets an alias exactly like on the console

### **Control: set <cvar> <value>**

Sets a console variable exactly as on the console

### **Control: fset <cvar> <value> <flags>**

Calls a console variable full set, which allows the menu to set certain control flags.

### **Control: cbuf <command>**

Executes a console command at the next opportune moment.

### **Control: config <config>**

Defines a configuration file to be run whenever the page is created or visited.

### **Control: exitcfg <config>**

Defines a configuration file to be conditionally run whenever a page is destroyed or left.

### **Control: timeout <value> <command>**

Defines the time holding on a page before executing a predetermined command.

-----

### **Layout: tint <name> <colour>**

Equates a colour to a name.

### **Layout: font**

Changes the current font on the page.

### **Layout: br**

A carriage return. These are reset to the top when a new layout command is encountered (eg normal, center)

### **Layout: hbr**

A carriage return. All following areas will appear below this.

### **Layout: center**

A layout command. All following areas will be centered on the screen.

### **Layout: normal**

A layout command. The areas will be placed from left to right until there is no more room, and then continue on the next line.

### **Layout: left**

A layout command. The following areas will be placed from left to right.

### **Layout: right**

A layout command. The following areas will be placed from right to left starting at the right hand side of the screen.

### **Layout: cursor <value>**

Defines the cursor to be used in this frame. Current valid values are 0 for no cursor and 1 for the standard cursor.

### **Layout: defaults**

Allows easy definition of a standard set of defaults.

### **Layout: translate <value>**

A way to translate the console variables that come down from the server into more meaningful names

### **Layout: emote <name> <text>**

Specifies a chat emote.

### **Layout: semote <name> <text>**

Specifies a system emote.

### **Layout: demote <name>**

Displays an emote

### **Layout: autoscroll <value>**

Defines the speed at which the page vertically scrolls

### **Layout: dmnames <list>**

Sets the list of names for the different deathmatch types

### **Layout: tab <value> [value]**

Defines a list of tab stops for layout

### **Layout: strings <sp>**

Includes a string package to be used on the page.

### **Layout: bghoul <model>**

This loads in a ghoul model as a background so as to allow other areas to be bolted to it.

### **Layout: backdrop**

Defines the background of the page.

---

### **Area: blank <width> <height>**

A blank area used for layout purposes

### **Area: hr**

Displays a horizontal rule.

### **Area: vbar**

Creates a vertical scroll bar down the right hand side of the screen.

### **Area: text <text>**

This creates an area containing a word that has properties (eg it is clickable).

### **Area: ctext <cvar>**

This creates an area containing the text contained in the console variable.

### **Area: image <image>**

This creates a area the size of the image loaded.

### **Area: list <list>**

This is the general workhorse area. You give the list a list of valid options to go through and a left mouse click will go forward through them, a right mouse click backwards.

### **Area: slider**

This creates a generic slider area, which can be horizontal or vertical. Used for dynamic scaling of console variables (eg gamma)

### **Area: ticker <text>**

Creates an item of scrolling text to the width you define.

### **Area: input**

Defines a text input box.

### **Area: setkey <action>**

Examines the following command and displays up to two keys that are bound to that command.

### **Area: popup <list>**

Defines a popup that comes up on a right mouse click.

### **Area: selection <image> <width> <height>**

A box which allows the selection of different types of items.

### **Area: ghoul <model>**

This loads and displays a ghoul 3d model.

### **Area: gpm <cvar> <cvar>**

This loads and displays a sequence of gpm files (player model description files) to allow the user to pick a player model.

### **Area: filebox <root> <base> <ext>**

Displays a list of files from a given directory from which the user can select one.

### **Area: filereq <root> <base> <ext>**

A basic file requester.

### **Area: loadbox**

Displays a list of savegame files with images, titles and dates

### **Area: serverbox**

Displays a list of servers which you can join

### **Area: serverdetail**

Displays the details of the selected server. The console variables returned are translated into more meaningful data.

### Area: players

Displays a list of players currently on a server

### Area: listfile <file>

Displays the contents of a maplist file, and allows selection of one item.

### Area: users

Displays a list of current residents of a chatroom

### Area: chat

A standard chat text display box.

### Area: rooms

Displays a list of valid chat rooms, from which you can select one to join.

## Control Keyword Details

### **Control: <stm>**

This defines the start of data to be parsed in an rmf file. There must be a </stm> somewhere later in the file for the file to be valid. The </stm> defines the end of the data to be parsed. Anything before the <stm> or after the </stm> will be ignored completely. There can only be one <stm> and one </stm> per rmf file.

Keywords:

**nopush** – this page is not pushed onto the page stack - it cannot be revisited.

**waitanim** – the page waits for the ghouls animation to finish before allowing any input. (E.g. on the animating main page)

**fragile** – Any keypress will instantly exit this screen. (E.g. On the start up animation)

**global** - disable the default scissoring to clip the contents of this page to its owning frame.

**command <command>** – Execute a command (normally to set up data on the page) before creating the page. (E.g. the game statistics page)

**resize <width> <height>** - Resize the current frame to a new width and height and centre on the screen. A resized menu can be locked to the position of the cursor by holding down the right mouse button anywhere inside it. (E.g. any requestor type menu).

### **Control: <frame name width height>**

Frames define the layout of a page. The “default” frame is the full size of the menu (which may have been resized), and successive frames are cut from the default frame or the named frame. All frames are scaled to the resolution so as the whole page fills the screen.

Keywords:

**page <name>** - Defines the page (rmf) file associated with this frame.

**cpage <cvar>** - The parser finds the contents of the console variable and uses that as the name of the page to load.

**cut <name>** - Defines the name of the frame from which this frame is cut.

**border <width> <line width> <line colour>** - This defines a border which fits inside the frame. Normally used to outline requesters and normally has no page attached to it.

**backfill <colour>** - Defines the background colour of the frame. Mainly used for frames with no page attached.



**Control: <key keyname command>**

When **keyname** is pressed when this page is active, **command** will be added to the console exec buffer. The keynames are the same as the bind names (E.g. A, B, ENTER, KP\_END etc.). There is also a special keyname “any” which refers to any keypress.

E.g. <key any “popmenu”> will pop the current menu off the menu stack if any key is pressed.

**Control: <ikey action command>**

When any key bound to **action** is pressed while the page is active, **command** will be added to the console exec buffer.

E.g. <ikey +attack “echo hello”>

If mouse1 and lctrl are bound to +attack, then when either of these keys are pressed, “hello” will be echoed to the console.

**Control: <ckey keyname cvar falsecommand truecommand>**

When **keyname** is pressed anywhere on the screen, **cvar** is checked, if it is zero, then **falsecommand** is added to the command buffer, if it is non zero, then **truecommand** is added to the command buffer.

E.g. <key enter console "play sound/menus/invalid.wav" "menuoff">

If enter is pressed and the console is enabled, then the menus will all be turned off. If the console is disabled (zero), then the invalid sound will play.

**Control: <comment text>**

Used to add comments to the rmf page. Completely ignored by everything.

**Control: <includecvar cvar>**

Includes the text contained in the console variable directly as source. The console variable text can have control sequences and layout keywords in it.

E.g.

```
<set menu_text “<center><font type title>Hello!<normal>”>
```

```
<includecvar menu_text>
```

**Control: <include page>**

Includes a different page of source into this page. This is a raw text inherit. A rmf extension is automatically added and the page must reside in the “base/menus” folder.

E.g. <include globe> will include “menus/globe.rmf” into the current page always.

**Control: <cinclude cvar page>**

Checks the value of **cvar**, and includes page if, and only if, the **cvar** is non zero.

E.g. <cinclude window\_avail m\_fullscreen\_toggle> will include “menus/m\_fullscreen\_toggle.rmf” into the current page if, and only if, the contents of the window\_avail cvar are non zero.

**Control: <cninclude cvar page>**

Only includes page if cvar has a value of zero.

E.g. `<cninclude no_won m_public_server>` will include “menus/m\_public\_server.rmf” if, and only if, no\_won is zero.

**Control: `<exinclude cvar page1 page2>`**

Checks the value of cvar and includes page1 if it is zero, otherwise it includes page2.

E.g. `<exinclude mail_avail blank letter>` will include “menus/letter.rmf” if mail\_avail is non zero or “menus/blank.rmf” if mail\_avail is zero.

**Control: `<einclude cvar value page>`**

Includes page if the cvar exactly equals value.

E.g. `<einclude mapname sib1 sib1mission>` will include “menus/sib1mission.rmf” if the mapname is sib1.

**Control: `<alias command actions>`**

This works exactly like an alias on the console, it strings a list of commands together and gives them simple name.

E.g. `<alias taunt “say Come get some loser!”>`

**Control: `<set cvar value>`**

Sets cvar to value immediately. It does not wait for the page to be loaded, it is evaluated when the command is parsed.

E.g. `<set gl_mode 3>` will set the screen mode to 640x480

**Control: `<fset cvar value type>`**

Sets cvar to value with type [u]serinfo, [s]erverinfo, [w]eapon, [a]mmo, [I]tem or [m]isc. Console variables with these attributes also automatically get saved to a save game file.

E.g. `<set knife 6 w>` will set the cvar “knife” to 6 and define it as a weapon type console variable.

**Control: `<cbuf command>`**

Executes a command on the console after the page is loaded.

E.g. `<cbuf “exec default”>` will exec default.cfg at the next opportune moment.

**Control: `<config config>`**

This defines a config file that is run whenever the page is created or revisited.

E.g. `<config menus/video>` will exec menus/video.cfg whenever the page is visited.

**Control: `<exitcfg config>`**

This defines a config file to be run whenever a page is destroyed or left.

E.g. `<exitcfg applysettings>` will exec applysettings.cfg whenever the page is left.

**Control: <timeout seconds command>**

This defines a command to be run after a certain number of seconds have expired.

E.g. <timeout 5.0 “menu intro”> will start up the intro menu after 5 seconds have passed since creation of the original page.

## Layout Keyword Details

### Layout: tint

Equates a 32 bit RGBA longword colour to a name. The format is little endian, so the components appear in the order ABGR. It is a layer of abstraction between raw numbers and names. The colour name can be used anywhere a colour can be, and a colour does not have to be named. For example, both “red” and “0xff0000ff” are both perfectly valid and usable in any situation. The default colour names are defined in “menus/tints.rmf”.

E.g.

```
<tint red 0xff0000ff>
<tint blue 0xffff0000>
<tint gray 0xff808080>
<tint darken 0x60000000>
<tint shadeblue 0x60ff0000>
```

### Layout: font

Changes the current font on the page. All fonts contain the same characters as the console font and based of the diatrical font at the end of this document.

Keywords:

**type** – this can be either title, medium or small. View the different fonts by type “menu fonts” at the console.

**tint** – defines the default colour for all following text

**atint** - defines the default hilite colour for all following text.

E.g. <font type medium tint normaltext atint hilitetext>

### Layout: br

A carriage return. This moves the current “next area position” down the height of the current font and to the left of the frame. If this is not a valid position, it will continue moving down until it is. Whenever the layout control is changed (i.e. a <right> or <center> keyword is encountered), the “next area postion” is set to the top of the page.

E.g. <br>

### Layout: hbr

A carriage return. This works exactly the same as <br> with the exception that the “next area position” is not reset to the top on a layout change. All following areas appear below the previous area.

E.g. <hbr>

**Layout: center**

A layout command which forces all following areas to appear centred on the screen and applies a carriage return (I admit that this is not ideal). If you wish to centre a line with spaces in it, then place the phrase in quotes.

E.g. <center>

**Layout: normal**

A layout command which places all following areas to the right of the previous area until no more will fit, and then a carriage return is applied. This is the default layout state, all areas appearing from left to right just like on a page of text.

E.g. <normal>

**Layout: left**

A layout command which places all the following areas from the left of the page going right. The only difference between this and <normal> is that <left> starts from the next available place on the left of the page and not the current “next area position”.

E.g. <left>

**Layout: right**

A layout command which places the following areas at the right of the page working left.

E.g. <right>

**Layout: cursor**

Defines the cursor to be used in this frame. Current valid values are 0 for no cursor and 1 for the standard cursor.

E.g. <cursor 0> would turn off the cursor in this frame.

**Layout: defaults**

Allows easy definition of a standard set of defaults, and allows configuration of page wide items.

Keywords:

**noborder** – All following areas are created without a border unless specifically requested.

**border <width> <line width> <colour>** – All following areas are created with the border defined here.

**tiptint <colour>** – Defines the colour of the text in the tooltip popups.

**tipbtint <colour>** – Defines the colour of the background of the tooltip popups. The light edge is this colour 33% brighter, the dark edge is this colour 33% darker.

**tipfont <font>** – Defines the font to be used in the tooltips.

**Layout: translate**

A way to translate the serverinfo data that comes down from the server to the client. These include maxclients, sv\_violence and fraglimit. The only required parameter to the translate is an integer of the current deathmatch type, this is so certain data can be ignored for certain deathmatch types (e.g. we have no interest in the CTF teams in standard deathmatch). If the integer is 0, then the console variables are translated for all deathmatch types. All console variables are listed in the order they are in the translate command. The bits are listed in a comma separated string (e.g. “Spinning Weapons,No Armour,No Health,Force Join”) – bit 0 refers to “Spinning Weapons”, bit 1 to “No Armour” and so on.

Keywords:

**conv <cvar> <name>** – converts a console variable name into real text. The console variable value is displayed as normal.

**convbits <cvar> <name> <list>** – converts each bit of the console variable’s bitfield into text.

E.g.

These are translated for all deathmatch types

```
<translate 0
conv "maxclients" "Max Players"
conv "fraglimit" "Frag Limit"
>
```

These are translated if the deathmatch type is 4 (CTF)

```
<translate 4
conv "ctf_team_red" "Red Team"
conv "ctf_team_blue" "Blue Team"
>
```

Any bit set in sv\_violence will cause that index in the comma separated list to be displayed.

```
<translate 0
convbits "sv_violence" "Violence" "Blood,Sever,Gibs,Damage,Textures"
>
```

### **Layout: emote**

Specifies a chat emote. In the simple chat client, typing “/laugh” (for example) will bring up a predefined message. All these messages are defined in “menus/chatmessages.rmf” with a series of emote commands. There are two control sequences that can appear in any order (or not at all) in the emote. %n refers to the person typing the message and %p is the parameter (or list of parameters) supplied on the command line. If %p is used but not given on the command line, “everyone” will be used. All emotes appear in orange.

E.g.

```
<emote "laugh" "%n laughs heartily at %p">
```

### **Layout: semote**

Specifies a system emote. These are messages that can only be used internal to the menu system, they are used to display system messages (e.g. “Fred has left the room”). All system emotes appear in yellow.

E.g. <semote "left" "%n has left the room">

### **Layout: demote**

Displays an emote definition. Used on the chat help screen to show available emotes. %n is replaced by ??? and %p is replaced by XXX. The current font type and tint are used.

E.g. <demote laugh>

### **Layout: autoscroll**

Defines the speed at which the page vertically scrolls. Used on the credits screen to make the credits scroll slowly by. The lower the number, the slower the scroll speed.

E.g. <autoscroll 1.0>

### **Layout: dmnames**

Sets the list of names for the different deathmatch types. Used in the server browse menu to convert the deathmatch value to a name. So when the parser needs to display deathmatch type 5, it displays the name “Realistic” instead.

E.g. <dmnames "Standard,Assassin,Arsenal,CTF,Realistic">

**Layout: tab**

Defines a list of tab stops for layout in pixels. Normally used for one tab stop, but can define an unlimited amount. For example, this is used on the key binding screen. One tab is defined (<tab 200>) and then the following areas have a keyword tab inside them, which tells them to use that tab stop. Another example is in the server list menu, which has multiple columns. Many tab stops are defined (<tab 180 224 304 432 480 512>) and the serverbox area uses these to space out the different columns inside it. All tab stops are scaled to the resolution, and the rendering to a tab stop is clipped to the following one.

E.g. <tab 100 200 300>

**Layout: strings**

Includes a string package to be used on the page. Used for localisation. The text of type “^MENU\_GENERIC\_HELLO^” grabs the string referenced by token “HELLO” (according to the current language) from the string package “MENU\_GENERIC”. The string package needs to be loaded before any token can be grabbed from it. This command allows this.

E.g. <strings MENU\_GENERIC>

**Layout: bghoul**

This loads in a ghoulish model as a background so as to allow other areas to be bolted to it. See the definition of the ghoulish area for details on this.

**Layout: backdrop**

Defines the background of the page. This can be formatted in many ways.

Keywords:

**tile** – tiles the following image to fill the frame (or the whole screen if the global flag is set)

**stretch** – stretches the following image to fill the frame.

**center** – centres the following image in the frame. Does not scale at all.

**left** – puts the following image flush to the left of the screen

**right** – puts the following image flush to the right of the screen.

**bgcolor** – defines the background colour to be used.

E.g. <backdrop “center misc/logo” bgcolor green> would centre “base/pics/menus/misc/logo.m32” on the frame and fill the rest of the screen green.

## Area Keyword Details

Areas define the actual content of the screen. They are rectangular and lay themselves out automatically depending on the current control status. Each area is designed to have a good set of defaults, or at least make a good guess as to what it should be to minimise the number of parameters that need to be passed and/or parsed.

The following keywords can be used with all area types, although some have no meaning, are redundant or it would be silly to use them with some area types.

Faded out if unalterable.

Key:

Keywords:

**key <key> <command>** - when this area is highlighted, and key is pressed, then command is added to the exec buffer.

**ckey <key> <cvar> <false command> <true command>** - when this area is highlighted, and key is pressed, if cvar contains zero, then false command is added to the exec buffer otherwise true command is added.

**ikkey <action> <command>** - if any key bound to action is pressed while this area is highlighted, then command is added to the exec buffer.

**tint <colour>** - the area will have the tint of the colour. E.g. “tint blue” will filter out all red and green components

**atint <colour>** - when this area is highlighted (ie the cursor is over the area) the area will be tinted by colour.

**btint <colour>** - some areas require/can have a backfill colour – this is it.

**ctint <colour>** - some areas need additional highlighting/differentiating colours – this is one of them.

**dtint <colour>** - some areas need additional highlighting/differentiating colours – this is the other.

**noshade** – some areas automatically shade different parts of their components. noshade disables this feature.

**noscale** – bolt offsets are automatically scaled to the resolution. Noscale disables this feature.

**border <width> <line width> <line colour>** - adds a border to the current area. The border is width wide, and in the centre is a line of colour line colour that is line width wide.

**width <value>** - overrides the default width of the area. Required for a very few areas.

**height <value>** - overrides the default height of the area. Required for some areas.

**cvar <cvar>** - associates a cvar with this area. Used to pass data to and from areas.

**cvari <cvar>** - associates a console variable that can only have integer values to this area.

**inc <value>** - specifies the step value to be used when keyboard shortcuts are used.

**mod <value>** - specifies the modulus of the area - the value at which the console variable associated with the area will wrap around.

**tip <text>** - specifies the tooltip for this area.

**xoff <value>** - specifies the horizontal offset from its defined position of this area. Normally used with bolted areas.

**yoff <value>** - as above, but vertical offset.

**tab** - use a tab stop when laying out this area.

**noborder** – if a border is defined in the defaults, then this turns off borders for this area.

**bolt <bolt>** - bolts the current area to a ghoult bolt in the most recently defined ghoult model. (more bolting info)

**bbolt <bolt>** - bolts the current area to the ghoult bolt defined in the ghoult model used for the background (bghoult) model.

**align <align>** - either left, center or right. Used as an additional layout tool to override the current page settings.

### **Area: blank <width> <height>**

A blank area used for layout and collision purposes. For example, if you need a margin down the left hand side, just define a blank area of the margins width, very high and place on the left hand side of the screen. The other major use of blank areas is to define collision spots on ghoult models. The ghoult model will have a bolt in the correct place but no accurate way of defining a box to click on. Attaching a blank area to the bolt is a way to do this.

E.g.

<blank 16 1024> would create a 16 wide margin

<blank 32 32 bolt null\_button key mouse1 “popmenu”> would attach a 32x32 clickable area to the null\_button bolt on the most recently defined ghoult model.

### **Area: hr**

This command does a <hbr> command, displays a horizontal rule, and then another <hbr>. This is a good way of separating out different topics on the same page. It defaults to white and the full width of the frame.

E.g.

`<hr width 200 tint gray>`

#### **Area: vbar**

Creates a vertical scroll bar down the right hand side of the screen. It automatically sizes to the size and position of the page. An area with the variable keyword defined can change size – this is how things like the server list dynamically adjust the vbar properties.

E.g. `<vbar tint vbargray>`

#### **Area: text <text>**

This creates an area containing a word that has properties (eg it is clickable). It uses the current font, font colour and highlight colour unless overridden. Regular words typed into the page adopt format themselves to the dimensions of the page according to the layout controls, however, they cannot be made clickable. The text area enables this.

E.g. `<text “DISCONNECT” key mouse1 “disconnect”>`

#### **Area: ctext <cvar>**

This creates an area containing the text contained in the console variable. It uses the current font, font colour and highlight colour unless overridden. This works in very much the same way as text, with the exception that the text is indirected through a console variable.

E.g. Your GL Driver is `<ctext gl_driver>`

#### **Area: image <image>**

This creates an area the size of the image loaded.

Keywords:

**abs <x> <y>** - Sets the absolute position of the image avoiding all the automatic layout (not recommended)

**alt <image>** - Sets the alternate highlight image of this area. When the cursor moves over this area, the original image is changed to this one. This image will scale to the dimensions of the original.

**hflip** - Flips the image horizontally.

**vflip** - Flips the image vertically.

**text <text> <xoff> <yoff>** - adds an overlay of text at offset xoff, yoff from the top left hand corner of the image.

**cvartext <cvar> <xoff> <yoff>** - adds an overlay of the text contained in console variable at offset xoff, yoff from the top left of the image.

**scale <value>** - scales the whole image in both dimensions by this value

**transient** - this keyword instructs the image to be reuploaded every frame the contrast, brightness, gamma or intensity is changed. This is used to display gamma bars.

**setimage <cvar>** - the image displayed is named in the console variable string. Used for the crosshair.

#### **Area: list <list>**

This is the general workhorse area. You give the list a list of valid options to go through and a left mouse click will go forward through them, a right mouse click backwards. If no matching string is given, then the index of the item is the value put into the console variable. A caveat to watch out for is having two matching values of the same value.

Keywords:

**atext <text>** - defines text to be placed before the item.

**match <list>** - defines a parallel list of values that the items in the original list refer to.

**bitmask <bit number>** - works on the bit number of the console variable value rather than the whole console variable. Used for deathmatch flags. More than 2 items in the list could cause undefined results.



**files <root> <base> <ext>** - grabs the list of files from “root/base.ext” and allows the user to select through the files. Only the base part of the filename is displayed.

E.g.

<list “640x480,800x600,1024x768” match “3,4,6” cvar gl\_mode> would be awkward as every time you click on this box, the video system would be restarted.

<list “56k,Cable,DSL,T1” match “3500,10000,10000,20000” cvar rate> will not work past the Cable option because it will set rate to 10000, relookup the value and think it is Cable.

#### **Area: slider**

This creates a generic slider area, which defaults to horizontal, but can be vertical. The max (far right or top) must be greater than the min (far left or bottom). The default min is 0.0f and the default max is 1.0f. While highlighted, pressing the arrow keys will modify the slider value by the step value (which defaults to 0.1f). Buttons at either end of the slider bar are created automatically and also modify the console variable by the step value.

Keywords:

**min <value>** - the value the console variable will be set to when the button is at the far left of the slider. Defaults to 0.0f.

**max <value>** - the value the console variable will be set to when the button is at the far right of the slider. Defaults to 1.0f.

**step <value>** - modifies the step value from the default 0.1f.

**vertical** – makes the slider bar a vertical one.

**horizontal** - make the slider bar a horizontal one (default)

**bar <image>** - overrides the default bar graphic to your custom one.

**button <image>** - overrides the default button graphic.

**cap <image>** - overrides the default end cap graphic.

E.g. <slider cvar vid\_gamma> will create a horizontal slider bar ranging from 0.0f to 1.0f and directly modify the vid\_gamma console variable.

#### **Area: ticker <text>**

Creates an item of scrolling text to the width you define. This area defaults to the height of the current font and the width of the page.

Keywords:

**speed** - the speed at which the text scrolls. The higher the number, the faster the scroll.

**delay** – the time to wait before starting the text scrolling. This is to allow animations to finish before the text starts to scroll.

#### **Area: input**

Defines a text input box. Used to input text into the game.

Keywords:

**global**- any key press on the screen will go to this box. The area does not have to be highlighted to accept text.

**hidden**- all entered text comes up as \*. Used to enter passwords.

**maxchars** – the input box will allow this many characters and then stop. The text will scroll to fit and clip off the left.

**history** - this input box remembers all strings typed into it. Used for chat history.

#### **Area: popup <list>**

Defines a popup that comes up on a right mouse click.

This area is currently unused and not recommended.

**Area: setkey <action>**

Examines **action** and displays up to two keys that are bound to that action. When this item is clicked on, the menu system goes into bind mode and the next key that is pressed will be bound to that action. If no keys are bound to the action, then ??? will be displayed. While in bind mode, the area fades in and out.

Keywords:

**atext <text>** – The text associated with this area. Appears to the left of the area.

**talign <align>** - aligns the text within the area.

E.g. <setkey "+attack" atext "Attack : ">

**Area: selection <image> <width> <height>**

A box which allows the selection of different types of items. There can be a maximum of four selection areas per page (weapons, ammo, items and miscellaneous). If a console variable with the weapon flag is selected, then it will be added to the selection box marked weapon. When the selection box is empty, image is repeated to define the number of empty slots. Width must be a multiple of the image width. If an attempt is made to add an item that requires too many slots then the invalid sound will play. The item descriptions are defined in cvar\_info console variables (e.g. knife\_info, sniper\_info). These are all defines in "menus/reset.cfg". E.g. set sniper\_info "weapons/w\_sniper,1,a556,2" means that the sniper uses icon "pics/menus/weapons/w\_sniper.m32", when it is selected you get one of them, it uses a556 ammunition and takes up 2 slots. E.g. set a556\_info "items/a\_556,120,none,0" uses icon "pics/menus/items/a\_556.m32", you select 120 at once, it has no ammunition and takes up 0 slots.

Keywords:

**weapons** – defines this selection area to take weapons.

**items** – defines this selection area to take items.

**ammo** – displays selected ammo.

**misc** - displays selected miscellaneous console variables.

**Area: ghoul <model>**

This loads and displays a ghoul 3d model. The model is rendered in orthographic projection and the main way of displaying background content in the menus. Although the parser makes a best guess at the size of the model, supplying your own width and height is recommended. All input is ignored until all animations have finished playing unless the page is marked fragile.

Keywords:

**yaw <cvar>** - sets the models yaw to the value from **cvar**

**pitch <cvar>** - sets the models pitch to the value from **cvar**

**roll <cvar>** - sets the models roll to the value from **cvar**

**scale <value>** - scales the model by value. A scale of 2.0 will double the size of the model.

**oneshot** - Only play the model animation once, stop at the end.

**control** - Make this model the controlling one. Used for pages with multiple animating models.

**time <value>** - Sets the models animation to time **value**.

**animate** – Informs the parser that this model animates.

**rotate <roll> <yaw> <pitch>** - sets the angular velocity of the model.

**rotation <roll> <yaw> <pitch>** - sets the initial rotation of the model.

**mousex <cvar> <value>** - allows the user to lock horizontal movement of the mouse while over the area. This movement is scaled by **value**.

**mousey <cvar> <value>** - allows the user to lock vertical movement of the mouse while over the area. This movement is scaled by **value**.

**gbolt <bolt1> <bolt2>** - bolts **bolt2** of the model to **bolt1** of the most recently defined ghoul model.

**bgbolt <bolt1> <bolt2>** - bolts **bolt2** of the model to **bolt1** of the background ghoul model.

**end <command>** - when the animation ends, add **command** to the exec buffer.

**Area: gpm <skin cvar> <team cvar>**

This loads and displays a sequence of gpm files (player model description files) to allow the user to pick a player model.

Keywords:

**Cvar2 <cvar>** - a console variable used to store the name of the team icon.

**Area: filebox <root> <base> <ext>**

Displays a list of files from a given directory from which the user can select one. Only the **base** part of the name is displayed, the **root** and **ext** are removed. Wildcards are allowed.

Keywords:

**backfill <colour>** - defines the background fill colour of the area.

**sort <sort>** - Defines the sort criteria of the files to be displayed. Can be "name", "rname", "size", "rsize", "time" or "rtime". The criteria starting with 'r' are from lowest to highest, rather than the other way around.

**talign <align>** - Defines the layout of the filenames inside the area.

E.g. <filebox "menus" "\*" "rmf" cvar filename> would set the console variable "filename" to the name of the menu file selected.

**Area: filereq <root> <base> <ext>**

A basic file requester.

Keywords:

**backfill <colour>** - defines the background fill colour of the area.

**sort <sort>** - see **filebox**.

**talign <sort>** - Defines the layout of the text inside the requestor.

**Area: loadbox**

Displays a list of save game files with images, titles and dates. The filespec is "user/save/\*.sof", and the default sort criteria is "rtime" (the newest files appear at the top). The parser examines the width of the area and tries to place as many load games per line as possible (i.e. more at higher resolutions).

Keywords:

**backfill <colour>** - Defines the background fill colour of the area.

**sort <sort>** - see **filebox**

**saving** - means that this area is used for saving games on this page. The autosave file is not displayed and the area will not be created if the user is not in a game.

**spacing <value>** - The number of pixels bordering each save game image.

**Area: serverbox**

Displays a list of servers which you can join. Clicking on a server will extract more detailed information which will appear in the serverdetail area.

Keywords:

**column <index> <name>** - Defines the name of the column and the index of the data in the serverinfo packet. Valid indices are 0 (host name), 1 (map name), 2 (players), 3 (game type) or 4 (ping). Columns can appear in any order (or not at all) and are spaced out according to the tab settings.

**dmnames <list>** - Defines which game type numbers are which game type names.

**hostname <cvar>** - The console variable to set to the host name of the currently selected server.

**address <cvar>** - The console variable to set to the address (either IP or IPX) of the currently selected server.

**Area: serverdetail**

Displays the details of the selected server. The console variables returned are translated into more meaningful data. Examine the **translate** command to see how this works in detail.

Keywords:

**title <list>** - Defines the titles of the columns. Normally “Key” and “Value”. A horizontal bar is placed after the title and the parsed fields appear after that.

**Area: players**

Displays a list of players currently on a server, along with their current frag count and ping.

Keywords:

**title <list>** - Exactly like the **serverdetail** title.

**Area: listfile <cvar>**

Displays the contents of a maplist file, and allows selection of one item.

**Area: users**

Displays a list of current residents of a chatroom. This list is updated automatically as users enter and leave the room.

**Area: chat**

A standard chat text display box which displays all the text currently in the chat buffer. You can scroll back upto a 100 lines of text.

**Area: rooms**

Displays a list of valid chat rooms, from which you can select one to join. This list is updated whenever the “won\_list\_rooms” command is used.

## Font Base

	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~	□
€	□	,	f	„	...	†	‡	ˆ	%	Š	€	£	□	□	□	□	‘	’	“	”	▪	-	—	~	™	š	>	œ	□	□	ÿ
	ı	đ	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

