

Contents

| | | |
|----|---|----|
| 1 | xit_loadtiles — <i>Load an IFF with compressed tiles.</i> | 4 |
| 2 | xit_freetiles — <i>Free a tileset.</i> | 5 |
| 3 | xit_draw — <i>Blit from a tileset into the specified SDL_Surface.</i> | 6 |
| 4 | xit_qdraw — <i>Blit to default surface from default tileset</i> | 7 |
| 5 | view_new — <i>Allocate a small chunk of display memory with back store.</i> | 8 |
| 6 | view_free — <i>Deallocate an xiview and its surface(s).</i> | 9 |
| 7 | view_show — <i>Blit view image to surface it is created for.</i> | 10 |
| 8 | view_hide — <i>Restore the area behind a view from its backing store.</i> | 11 |
| 9 | xi_init — <i>Initialise Xiqua and all SDL subsystems.</i> | 12 |
| 10 | xi_inittag — <i>This function gets called by xi_init() when initialising.</i> | 14 |
| 11 | xi_quit — <i>Free the xiglob structure and all its related data, then release all SDL resources.</i> | 15 |
| 12 | xi_main — <i>Update mouse cursor (if using multicolour), and handle any input, calling user-specified hooks.</i> | 16 |
| 13 | net_create_server — <i>Create a new select() server.</i> | 17 |
| 14 | net_free_server — <i>Release all resources related to a Xiqua select() server.</i> | 18 |
| 15 | net_process — <i>Handle a select() server's incoming and outgoing data.</i> . | 19 |
| 16 | cfg_findnext — <i>Find a named section starting at a specific section.</i> | 20 |
| 17 | cfg_findsection — <i>Find a named section in a list.</i> | 21 |
| 18 | cfg_freesection — <i>Free a preferences section and its variables.</i> | 22 |
| 19 | cfg_freevar — <i>Free a variable and its strings.</i> | 23 |
| 20 | cfg_isbool — <i>Internal function for cfg_loadprefs().</i> | 24 |
| 21 | cfg_loadprefs — <i>Load preferences from an INI-style file.</i> | 25 |
| 22 | cfg_newsection — <i>Allocate a new preferences section.</i> | 27 |
| 23 | cfg_saveprefs — <i>Save a List of preferences sections to file.</i> | 28 |
| 24 | xi_seterror — <i>A generic way of setting error strings.</i> | 29 |
| 25 | file_getsize — <i>Stand-alone routine to get a file's size.</i> | 30 |
| 26 | file_new — <i>Open an autofile.</i> | 31 |
| 27 | file_free — <i>Close autofile and free all buffers.</i> | 32 |
| 28 | file_initbuf — <i>Allocate a buffer of the given size.</i> | 33 |
| 29 | file_load — <i>Load an entire file into memory.</i> | 34 |
| 30 | file_seek — <i>Seek-wrapper for SEEK_SET functionality.</i> | 35 |
| 31 | file_scan — <i>Seek-wrapper for SEEK_CUR functionality.</i> | 36 |
| 32 | file_read — <i>Read a number of bytes into the buffer from an autofile.</i> ... | 37 |
| 33 | file_write — <i>Write from an autofile's buffer.</i> | 38 |
| 34 | file_readhook — <i>Set an autofile's read hook pointer.</i> | 39 |
| 35 | file_writehook — <i>Set an autofile's write hook pointer.</i> | 40 |
| 36 | iff_close — <i>Closes an IFF.</i> | 41 |
| 37 | iff_correctchunk — <i>Correct and pad odd-sized IFF chunks.</i> | 42 |
| 38 | iff_endchunk — <i>Finishes writing a chunk.</i> | 43 |
| 39 | iff_new — <i>Creates a new IFF.</i> | 44 |
| 40 | iff_newchunk — <i>Start on a new chunk in an IFF.</i> | 45 |
| 41 | iff_writetechunkdata — <i>Write data to an IFF where you have just started a chunk.</i> | 46 |

| | | |
|----|--|----|
| 42 | list_add — <i>Add a Node to a List.</i> | 47 |
| 43 | list_addhead — <i>Add a Node to the top of a List.</i> | 48 |
| 44 | list_delete — <i>Remove a Node from a List and free data.</i> | 49 |
| 45 | list_free — <i>Free a List and all its attached Node structures.</i> | 50 |
| 46 | list_getnode — <i>Return a Node at a specific index position.</i> | 51 |
| 47 | list_insert — <i>Insert one Node after another given Node.</i> | 52 |
| 48 | list_makecircular — <i>Make a List circular (first and last Node points to eachother).</i> | 53 |
| 49 | list_new — <i>Create a new List and set the destructor.</i> | 54 |
| 50 | list_newitem — <i>Allocates a new Node structure.</i> | 55 |
| 51 | list_remove — <i>Remove a Node from a List without freeing any data.</i> ... | 56 |
| 52 | MD5Init — <i>Start MD5 accumulation.</i> | 57 |
| 53 | MD5Update — <i>Update context to reflect the concatenation of another buffer full of bytes.</i> | 58 |
| 54 | MD5Final — <i>Final wrapup.</i> | 59 |
| 55 | MD5Sum — <i>MD5-checksum a buffer.</i> | 60 |
| 56 | MD5Ascii — <i>Make a printable version of the MD5 sum.</i> | 61 |
| 57 | node_end — <i>Return the last Node in a circular List.</i> | 62 |
| 58 | node_findbycontents — <i>Return a Node with data partially containing the entire specified string.</i> | 63 |
| 59 | node_findbyname — <i>Return a Node with data containing the specified string.</i> | 64 |
| 60 | node_makecircular — <i>Make a chain of nodes circular (first and last Node points to eachother).</i> | 65 |
| 61 | node_start — <i>Return the first Node in a circular List.</i> | 66 |
| 62 | strfind — <i>Looks for a string within another string.</i> | 67 |
| 63 | strisnum — <i>Check if a string is all numbers and whitespace.</i> | 68 |
| 64 | strlower — <i>Turn a string into all-lowercase characters.</i> | 69 |
| 65 | str_tokenise — <i>Create a tokeniser object from a string.</i> | 70 |
| 66 | str_freetoken — <i>Free a tokeniser object.</i> | 71 |
| 67 | strupper — <i>Turn a string into all-uppercase characters.</i> | 72 |
| 68 | tag_alloclist — <i>Allocate a tag array big enough for numtags items.</i> | 73 |
| 69 | tag_finditem — <i>Look for a tag identifier in a taglist, and return a pointer to the tagitem.</i> | 74 |
| 70 | tag_freelist — <i>Free a tagarray previously created with tag_alloclist().</i> .. | 75 |
| 71 | tag_getdata — <i>Find a tagitem's data by tag ID.</i> | 76 |
| 72 | tag_next — <i>Get next tagitem in the array.</i> | 77 |
| 73 | autofile — <i>Autofile structure.</i> | 78 |
| 74 | xi_textwidth — <i>Calculate how many pixels wide a string printed with a fixed-width xfont will be.</i> | 82 |
| 75 | MAKE_ID — <i>Quick macro to make a ulong of four characters.</i> | 83 |
| 76 | IFFHandle — <i>Handle returned by iff_new() and iff_open() calls.</i> | 84 |
| 77 | Node — <i>A simple Node.</i> | 87 |
| 78 | List — <i>Linked list structure.</i> | 89 |
| 79 | list_nodemakecircular — <i>Wrapper for node_makecircular().</i> | 91 |
| 80 | list_nodestart — <i>Wrapper for node_start().</i> | 92 |
| 81 | list_nodeend — <i>Wrapper for node_end().</i> | 93 |
| 82 | MD5Context — <i>MD5 context for the Rivest/Plumb MD5 checksumming routines.</i> | 94 |
| 83 | preferences — <i>A preferences section.</i> | 95 |

Contents

| | | |
|----|---|-----|
| 84 | variable — <i>A variable.</i> | 97 |
| 85 | xiview — <i>This is a view, for use as sprites, message boxes, or anything else that might need to store the rectangle it is blitted over.</i> | 100 |
| 86 | Xiqua questions and answers. — | 105 |
| 87 | xiglob — <i>Global structure for Xiqua's automatic handling of miscellaneous data</i> | 108 |
| 88 | strtoken — <i>The token structure.</i> | 114 |
| 89 | tagitem — <i>A tagitem.</i> | 116 |
| 90 | XLVARARG — <i>Macro for vararg taghandling.</i> | 117 |
| 91 | XLVOIDARG — <i>Macro for vararg taghandling.</i> | 118 |
| 92 | xitiles — <i>A tileset in-memory representation.</i> | 119 |
| 93 | xit_setdefaults — <i>Set default screen and tileset</i> | 120 |

1

`xitiles* xit_loadtiles (char* name)`

Load an IFF with compressed tiles.

Load an IFF with compressed tiles. The tiles have been previously compiled with the `tileset` program. A tileset contains one or more tiles, the size of all tiles (same size for all), the colour that is to be considered transparent (handled automatically by the blitting functions) and a couple of rectangles for more efficient blitting.

See Also: `xitiles`, `xit_draw()`, `xit_qdraw()`, `xit_freetiles()`
Author: Ronny Bangsund

2

`void xit_freetiles (xitiles* tiles)`

Free a tileset.

Free a tileset. You must deallocate each tileset before the program exits, of course. This function frees all lists, arrays and surfaces allocated in a tileset.

See Also: `xit_loadtiles()`, `tileset`

Author: Ronny Bangsund

3

```
void xit_draw (xtiles* tiles, ulong num, SDL_Surface* dest,  
              Uint16 x, Uint16 y)
```

Blit from a tileset into the specified SDL_Surface.

Blit from a tileset into the specified SDL_Surface. This is one of two calls available to display tiles in a set. The recommended method is to call `xit_setdefaults()` and use `xit_qdraw()` for all main blits, and `xit_draw()` only for blits that don't happen very often. Your program's structure may of course affect the validity of this method.

See Also: `xit_loadtiles()`, `tileset`
Author: Ronny Bangsund

4

`void xit_qdraw (ulong num, Uint16 x, Uint16 y)`

Blit to default surface from default tileset

Blit to default surface from default tileset Uses the defaults set by the most recent call to `xit_setdefaults()` to blit a specified tile image to an `SDL_Surface`.

See Also: `xit_loadtiles()`, `xit_setdefaults()`, `tileset`

Author: Ronny Bangsund

5

`xiview* view_new (ulong tag1, ...)`

Allocate a small chunk of display memory with back store.

Allocate a small chunk of display memory with back store. This structure can be used for sprites, cursors, message boxes or anything else that might need to display and keep its background for re-blitting later.

Accepted tags:

XIVT_FLAGS: The flags are **XIV_FILLBACK** to fill the created rectangle in `view->image` with the specified background colour, **XIV_NOSTORE** to not create any storage for the background area and **XIV_NOKEY** to avoid colourkey blitting with the background colour as key (entire surface is copied instead of background showing through).

XIVT_SCREEN: SDL surface to blit onto (need not be main screen).

XIVT_XPOS, XIVT_YPOS: Position in surface to be blitted to.

XIVT_WIDTH, XIVT_HEIGHT: Size of view and background storage, unless an image to be loaded is specified.

XIVT_BGCOL: It's recommended that you use `SDL_MapRGB()` from the destination surface's format to set this, if you need it.

XIVT_IMAGE: An SDL image supported file to load and use as `view->image`.

See Also: `view_free()`, `view_show()`, `view_hide()`

Author: Ronny Bangsund

6

`void view_free (xiview* view)`

Deallocate an xiview and its surface(s).

Deallocate an xiview and its surface(s). The view will be hidden upon destruction, restoring any stored background. Note that this also means its screen element must be valid until after destruction.

Parameters: `view` A view structure created with `view_new()`
See Also: `view_new()`, `view_show()`, `view_hide()`
Author: Ronny Bangsund

7

`void view_show (xiview* view)`

Blit view image to surface it is created for.

Blit view image to surface it is created for. If there is backing store allocated, the area at the view's position is copied to this backing store. If the view is already set as displayed through a previous call to this function, nothing happens at all.

Parameters: `view` A view structure created with `view_new()`
See Also: `view_new()`, `view_free()`, `view_hide()`
Author: Ronny Bangsund

8

`void view_hide (xiview* view)`

Restore the area behind a view from its backing store.

Restore the area behind a view from its backing store. If the view flags has the `XIV_NOSTORE` bit set, nothing happens.

Parameters: `view` A view structure created with `view_new()`

See Also: `view_new()`, `view_free()`, `view_show()`

Author: Ronny Bangsund

```
xiglob* xi_init (ulong tag1, ... )
```

Initialise Xigual and all SDL subsystems.

Initialise Xigual and all SDL subsystems. The tags supplied allow you to selectively initialise some SDL subsystems, create a mousepointer from an SDL image supported file and more.

To initialise all supported SDL libraries and create the xiglob structure, with an 800x600x16 display:

```
xi_init(XI_WIDTH, 800,
XI_HEIGHT, 600,
XI_DEPTH, 16, TAG_END);
```

This creates a mouse cursor, and tells Xigual to also handle the cursor, in addition to opening a typical display:

```
xi_init(XI_WIDTH, 800,
XI_HEIGHT, 600,
XI_DEPTH, 16,
XI_CURSORFILE, "cursor.png",
TAG_END);
```

<h2>Supported tags</h2>

<h3>Xigual settings and hooks</h3>

- **XI_MAINHOOK** A pointer to a function to be called at each input poll. See `xisetup.h/xi_main()` for information on the arguments passed.
- **XI_PREFS** Load this configuration file. This is currently only for convenience, but Xigual may start looking for a section of its own for various settings in the future (type of SDL audio/video drivers to use etc.).

<h3>SDL settings</h3>

- **XI_WIDTH, XI_HEIGHT** Screensize. Will be emulated if using an odd resolution, or if it's not available according to the XF86Config (for example, 320x240 is usually not defined as available for 16-bit resolutions).
- **XI_DEPTH** Bitplane depth (note that SDL may emulate this).
- **XI_SDLFLAGS** If you need to initialise threading, or only a few SDL subsystems, pass the replacement flags here. These will be used instead of the defaults, which is `SDL_INIT EVERYTHING`.

- **XL_SCRFLAGS** Xigual will default to opening the screen with flags `SDL_HWSURFACE` — `SDL_DOUBLEBUF`, so pass a different set of flags here if that isn't what you want. Note that many other Xigual functions also rely on hardware buffers for blits, and these can't be changed (yet).
- **XL_CURSOR** You may pass a pointer to an xiview with this tag. Xigual frees this structure on exit.
- **XL_CURSORFILE** This tag is for those who prefer to create a mouse cursor image with more colours. A view is created, and will be freed on exit, like **XL_CURSOR**.
- **XL_CURSORKEY** Pass a colourkey (as returned from `SDL_MapRGB()`) here if you also specified a filename to load a cursor from. If specifying a cursor directly, that view's key will be used.
- **XL_CURSOR_HOTX**, **XL_CURSOR_HOTY** This is the location of the "hotspot", the point on the cursor image that is considered the active (selecting) part. The image will be drawn at mouse coordinates minus these coordinates.

SDL_mixer settings

- **XL_MIXCHAN** Maximum number of total mixing channels. Defaults to eight.
- **XL_AUDIORATE** Bitrate (frequency) to use for mixed audio, if `SDL_mixer` is available. Defaults to 22050Hz.
- **XL_AUDIOCHAN** Defines type of audio for music - 1 for mono, 2 for stereo. See **XL_MAXCHAN**, if you're confused.
- **XL_AUDIOBUFSIZE** Size of audio buffers - too small or too big may cause stuttering or unexpected pauses on older systems and bad sound hardware. Note: Must be a multiple of 16.
- **XL_AUDIO_FMT** Audio format to use for audio mixing. May be one of:
 - **AUDIO_U8** Unsigned 8-bit.
 - **AUDIO_S8** Signed 8-bit.
 - **AUDIO_U16LSB** 16-bit unsigned (Intel format).
 - **AUDIO_S16LSB** 16-bit signed (Intel format). This is the default for WAV.
 - **AUDIO_U16**, **AUDIO_S16** Shorthand for the two Intel formats.
 - **AUDIO_U16MSB** Big-endian unsigned 16-bit.
 - **AUDIO_S16MSB** Big-endian signed 16-bit.
 - **AUDIO_U16SYS** 16-bit audio of whatever is the native byte-order.
 - **AUDIO_S16SYS** 16-bit signed audio of native byte-order.

See Also: `xi_inittag()`, `xi_main()`, `xi_quit()`, `SDL_Init()`,
`Mix_OpenAudio()`, `xiglob`

Author: Ronny Bangsund

10

`void xi_inittag (ulong tag1, ...)`

This function gets called by xi_init() when initialising.

This function gets called by xi_init() when initialising. You may pass similar tags as xi_init() to this function after setup to change resolution, audio mode or shut down subsystems.

See Also: xi_init(), xi_main(), SDL_Init(), Mix_OpenAudio(),
 xi_quit()

Author: Ronny Bangsund

11

void xi_quit ()

Free the xiglob structure and all its related data, then release all SDL resources.

Free the xiglob structure and all its related data, then release all SDL resources. This function replaces SDL_Quit(), so please check your atexit() calls.

See Also: xi_init(), xi_inittag(), xi_main(), Mix_CloseAudio(),
 SDL_Quit()

Author: Ronny Bangsund

12

`int xi_main ()`

Update mouse cursor (if using multicolour), and handle any input, calling user-specified hooks.

Update mouse cursor (if using multicolour), and handle any input, calling user-specified hooks. Use `xi_init()` or `xi_inittag()` to set an `XI_MAINHOOK` tag pointing to your input handler.

The hook will be called with a pointer to a keystate, a modstate and a mousestate (`Uint8 *`, `SDLMod` and `Uint8`, respectively), and two integers (`int`) with the current mouse coordinates.

Note: You'll have to use SDL's joystick functions yourself in this main loop to read any joysticks, as Xiqua currently has no code relating to joysticks. This might change if anyone ever wants even easier handling.

See Also: `xi_init()`, `xi_inittag()`, `xi_quit()`

Author: Ronny Bangsund

SelectServer* net_create_server (ulong tag1, ...)

Create a new select() server.

Create a new select() server. All parameters are passed in as a taglist. To create a simple server for no more than 8 connections, at port 1242, calling client_read() to process each incoming message:

```
SelectServer *server;
```

```
server = net_create_server(XN_NUMCONNECTIONS, 16, XN_TIMEOUT_S, 1,
XN_PORT, 1242, XN_ONREAD_FUNC, client_read, TAG_END);
```

The accepted tags for this function are as follows:

XN_NUMCONNECTIONS: Maximum number of connection nodes to make available.

XN_TIMEOUT_S: Timeout in seconds before select() returns.

XN_TIMEOUT_MS: As above, but milliseconds - 10ms being the lowest you can safely expect to work.

XN_PORT: Port to wait for connections on.

XN_LINGER: Number of seconds for sockets to linger after closing. If a socket doesn't have the linger setting activated, it may take up to 5 minutes before the operating system releases all resources properly.

XN_CONNECT_FUNC: Hook called right after connection.

XN_CLOSE_FUNC: Hook called just before shutting down a socket.

XN_ONREAD_FUNC: Hook called to read incoming messages. There is no default reader in libxinet yet.

XN_QUEUE: Maximum listen() queue. Defaults to 5.

Return Value: A pointer to a valid SelectServer structure, with sockets able to listen().

Parameters: tags Tags defining all server settings.

See Also: SelectServer, listen(), net_process(), xipacket, net_free_server()

Author: Ronny Bangsund

14

void net_free_server (SelectServer* server)

Release all resources related to a Xiqua select() server.

Release all resources related to a Xiqua select() server. This shuts down the sockets, calling each socket's onclose() hook if available, and cleaning up Win32 sockets if compiled for Win32.

See Also: SelectServer, net_create_server()

Author: Ronny Bangsund

15

`int net_process (SelectServer* server)`

Handle a select() server's incoming and outgoing data.

Handle a select() server's incoming and outgoing data. All user-supplied hooks will be called as required. Sockets may be shut down on error.

Return Value: Returnvalue of the select() call.

Parameters: server SelectServer structure created with net_create_server().

See Also: SelectServer, net_create_server(), xipacket, net_free_server()

Author: Ronny Bangsund

16

```
preferences* cfg_findnext (preferences* prefs, char* section-  
                             name)
```

Find a named section starting at a specific section.

Find a named section starting at a specific section. The given section is also compared against the name.

| | | | | | |
|----------------------|---|-------|---|-------------|-----------------------------|
| Return Value: | First matching section, or NULL. | | | | |
| Parameters: | <table><tr><td>prefs</td><td>Preferences section to start comparison at.</td></tr><tr><td>sectionname</td><td>An exact match to look for.</td></tr></table> | prefs | Preferences section to start comparison at. | sectionname | An exact match to look for. |
| prefs | Preferences section to start comparison at. | | | | |
| sectionname | An exact match to look for. | | | | |
| See Also: | preferences, cfg_findsection() | | | | |
| Author: | Ronny Bangsund, Shane O'Neill | | | | |

17

`preferences* cfg_findsection (List* list, char* sectionname)`

Find a named section in a list.

Find a named section in a list. The search starts at the head every time; use `cfg_findnext()` to search from the current node.

| | | | | | |
|--------------------------|--|-------------------|--|--------------------------|-----------------------------|
| Return Value: | First matching section, or NULL. | | | | |
| Parameters: | <table><tbody><tr><td><code>list</code></td><td>List of sections (preferences structures).</td></tr><tr><td><code>sectionname</code></td><td>An exact match to look for.</td></tr></tbody></table> | <code>list</code> | List of sections (preferences structures). | <code>sectionname</code> | An exact match to look for. |
| <code>list</code> | List of sections (preferences structures). | | | | |
| <code>sectionname</code> | An exact match to look for. | | | | |
| See Also: | <code>preferences</code> , <code>cfg_findnext()</code> | | | | |
| Author: | Ronny Bangsund | | | | |

18

void cfg_freesection (preferences* prefs)

Free a preferences section and its variables.

Free a preferences section and its variables. If you used `cfg_loadprefs()` to load the structures from a file, this function will be called as a destructor on each section. A simple `list_free()` on the List will deallocate all resources.

Parameters: `prefs` Preferences structure (section), as created by `cfg_newsection()`.

See Also: `preferences`, `cfg_loadprefs()`, `cfg_newsection()`

Author: Ronny Bangsund

19

`variable* cfg_freevar (variable* var)`

Free a variable and its strings.

Free a variable and its strings.

| | |
|----------------------|-------------------------------------|
| Return Value: | Next variable. |
| Parameters: | var Variable structure of any type. |
| See Also: | preferences, variable, cfg_newvar() |
| Author: | Ronny Bangsund |

20

int **cfg_isbool** (char* value)*Internal function for cfg_loadprefs().*

Internal function for `cfg_loadprefs()`. This will return *TRUE* if a string pointed to is a boolean value. There is a special case, however; if there is no string, it also returns *TRUE*. `cfg_loadprefs()` will pass the value from `strtok_r()` unchecked because I couldn't be arsed to do it any other way.

Return Value: `TRUE` if the string pointed to is a boolean value,
or `NULL`.

Parameters: `value` A string pointer (probably).

See Also: `variable`, `cfg_loadprefs()`

Author: Ronny Bangsund

21

```
extern List* cfg_loadprefs (ulong tag1, ... )
```

Load preferences from an INI-style file.

Load preferences from an INI-style file. The minimum configuration file accepted has one line with a variable name on it:

```
mybool
```

A standalone variable like this would be considered a boolean variable. When loaded into memory, you would be returned a List containing only one preferences section, with no name.

A larger file should be divided into sections, like this:

```
[main]
mybool
something = something else
```

The List returned from `cfg_loadprefs()` would now contain one preferences section named "main", with two variable - `mybool` is a boolean (set to TRUE), and the second variable would be named "something", a string variable set to "something else".

Whenever a new set of square brackets are encountered, the name between them is used in a new preferences section. All the following variables will be added to that until yet another section name is found.

Legal variable names are anything not containing whatever your system considers whitespace.

Accepted tags:

PREFS.FILENAME: Name of file to load. Required.

PREFS.STRINGS: Load all data as string variables.

PREFS.STRING.HOOK, PREFS.VALUE.HOOK: Normally, variables are tokenised to include the whole string (for string variables) or get numbers until `atoi()` returns (integers). Use these tags to specify an alternate function to handle the processing of the value. Useful if a string contains multiple filenames, or several numbers to process.

A variable structure will not be created, but the hook gets the name of the variable that otherwise would be created, along with the preferences section pointer and a string containing the rest of the line. You will then need to duplicate any strings passed with `strdup()`.

The hook functions are declared as such in the sourcecode:

```
void (*strhook)(struct preferences *section, char *varname, char *values);
void (*valhook)(struct preferences *section, char *varname, char *values);
```

See Also: preferences, variable, cfg_findsection(), cfg_findvar(),
list_free() strtok(), atoi(), strdup()
Author: Ronny Bangsund

22

```
preferences* cfg_newsection (ulong tag1, ... )
```

Allocate a new preferences section.

Allocate a new preferences section. This is used by `cfg_loadprefs()` when parsing the loaded file.

Accepted tags:

PREFS_NAME: A name for the section. This is optional, but strongly recommended to include.

PREFS_STRINGS: **TRUE** if you want to load the file as strings.

PREFS_LIST: List to add section to. Strongly recommended.

Return Value: A section (preferences structure) if all went well.

See Also: `List`, `preferences`, `variable`, `cfg_loadprefs()`, `cfg_newvar()`

Author: Ronny Bangsund

23

`int cfg_saveprefs (List* list, char* dirname, char* filename)`

Save a List of preferences sections to file.

Save a List of preferences sections to file. All sections will be visited in order, saving all variables without comments for each. If there is a previous file with the supplied name, it will be deleted first.

| | | |
|--------------------|-------------------------------|---|
| Parameters: | <code>list</code> | The list containing pointers to all your preferences sections. |
| | <code>dirname</code> | An optional directory to change to before saving. |
| | <code>filename</code> | The filename to save the structure as. |
| See Also: | <code>List</code> , | <code>preferences</code> , <code>variable</code> , <code>cfg_loadprefs()</code> , |
| | <code>cfg_newsection()</code> | |
| Author: | Ronny Bangsund | |

24

void xi_seterror (char* text)

A generic way of setting error strings.

A generic way of setting error strings. Sets the `xi_lasterror` variable to a user-specified string. The string is not duplicated, so it must stay alive for as long as the program exists (preferably), or at least until a new error string is set.

A near-future version of this call will check for a user-supplied hook (passed to `xi_init()` on startup) to call whenever a new error message is passed to it. This hook could be used to display error-message boxes or a way to solve problems.

Using GNU `gettext()` etc. for internationalised, or even just English, text strings is of course recommended.

Author: Ronny Bangsund

25

`size_t file_getsize (char* name)`

Stand-alone routine to get a file's size.

Stand-alone routine to get a file's size. If the file specified exists, its size is returned. Note that a file may be empty, in which case this fails miserably. This is fine for `file_new()` in Xiqua, however. Reading in an empty file is not necessary when you can just overwrite.

See Also: `fopen()`, `fseek()`, `fclose()`, `file_new()`
Author: Ronny Bangsund

26

autofile* file_new (char* filename, int write)*Open an autofile.*

Open an autofile. This function will either open an existing file, or create a new one for writing/append if the write parameter is set.

If a file exists and is opened for reading, af->size will contain its size.

Return Value: If the file exists or was created, return an autofile pointer.

Parameters: filename Name of a file to open/create.
write Non-NULL to create a file.

See Also: autofile, file_read(), file_write(), file_scan(), file_seek(), file_free()

Author: Ronny Bangsund

27

`void file_free (autofile* af)`

Close autofile and free all buffers.

Close autofile and free all buffers. Deallocates all structure members and the autofile after an `fclose()`.

See Also: `autofile`, `file_new()`, `fclose()`

Author: Ronny Bangsund

28

size_t file_initbuf (autofile* af, size_t s)*Allocate a buffer of the given size.*

Allocate a buffer of the given size.

Parameters:

- af Autofile to create the buffer in. Only one buffer is allowed.
- s Number of bytes. Most programs can get away with a few kiloytes. Bigger buffers tend to get written from an operating system's cache quicker, but it's still no guarantee. Larger buffers mean more efficient readoperations, though.

See Also: autofile, file_new(), file_read(), file_write()

Author: Ronny Bangsund

29**size_t file_load** (autofile* af)*Load an entire file into memory.*

Load an entire file into memory. Typically used on textfiles. If there is a readhook, it will be called on the entire file before returning. An extra byte will be allocated to ensure NULL termination.

See Also: autofile, file_new(), file_initbuf(), file_read(), file_free()

Author: Ronny Bangsund

30

`size_t file_seek (autofile* af, size_t pos)`

Seek-wrapper for SEEK_SET functionality.

Seek-wrapper for SEEK_SET functionality. Seek to a specified position from the beginning of the file. The pos member of struct autofile is updated.

Return Value: Current position.

Parameters:
af The autofile to seek in.
pos Exact position in file to go to. All reads will continue from there.

See Also: autofile, file_new()

Author: Ronny Bangsund

31

`size_t file_scan (autofile* af, size_t skip)`

Seek-wrapper for SEEK_CUR functionality.

Seek-wrapper for SEEK_CUR functionality. Seek from the current position to skip bytes later in the file. The pos member of struct autofile is updated.

Return Value: Current position.
Parameters: af The autofile to seek in.
skip Skip this many bytes towards the end of the file from current position.
See Also: autofile, file_new()
Author: Ronny Bangsund

32

size_t file_read (autofile* af, size_t s)*Read a number of bytes into the buffer from an autofile.*

Read a number of bytes into the buffer from an autofile. If the read operation works fine, the read hook will be called, if one exists. It will not read more than af->bufsize per read.

Return Value: Number of bytes actually read. This is also passed to the hook.

Parameters: af An autofile structure. The file will be opened if not already open.
s Number of bytes to read.

See Also: autofile, file_new(), file_readhook()

Author: Ronny Bangsund

33

size_t file_write (autofile* af, size_t s)*Write from an autofile's buffer.*

Write from an autofile's buffer. Write s bytes to file from the autofile's buffer, or the size of the buffer if smaller. That means it's safe to call with really large values, kids.

Return Value: Number of bytes actually written. Your hook may have gotten the passed-in number, which may differ in disk-full conditions.

Parameters: af An autofile structure. The file will be opened if not already open.
s Number of bytes to write.

See Also: autofile, file_new(), file_writehook()

Author: Ronny Bangsund

34

```
void file_readhook (autofile* af, void (*readhook)(char* buf,  
size_t len))
```

Set an autofile's read hook pointer.

Set an autofile's read hook pointer.

Parameters: af An autofile.
 readhook Your hook, or NULL to clear.

See Also: autofile, file_new(), file_read()

Author: Ronny Bangsund

35

```
void file_writehook (autofile* af, void (*writehook)(char* buf,  
size_t len))
```

Set an autofile's write hook pointer.

Set an autofile's write hook pointer.

Parameters: af An autofile.
 writehook Your hook, or NULL to clear.

See Also: autofile, file_new(), file_write()

Author: Ronny Bangsund

36

`void iff_close (IFFHandle* handle)`

Closes an IFF.

Closes an IFF. If it was opened with `iff_new()`, the final filesize is written to its header.

See Also: `iff_open()`, `iff_new()`

Author: Ronny Bangsund

37

void iff_correctchunk (IFFHandle* handle, IFFChunk* chunk)

Correct and pad odd-sized IFF chunks.

Correct and pad odd-sized IFF chunks. Used internally when a chunk is ended, or a new chunk is started.

See Also: `iff_newchunk()`, `iff_endchunk()`

Author: Ronny Bangsund

38**void iff_endchunk** (IFFHandle* handle)*Finishes writing a chunk.*

Finishes writing a chunk. This should be called when you are done with your iff_writechunkdata() calls. If the resulting chunksize is odd, an extra null-byte will be added at the end of the chunk, and the header will have its size set to the new value.

See Also: iff_new(), iff_writechunkdata()

Author: Ronny Bangsund

39**IFFHandle* iff_new** (char* name, ulong type)*Creates a new IFF.*

Creates a new IFF. Any existing file with the same name will be deleted. Note that header information is written in big-endian format. It's recommended to convert all numeric data to big-endian before saving if the data will be available on several different platforms. Intel x86 is little-endian, and PPC is big-endian.

Parameters: name Name of file to create.
 type IFF type.

See Also: iff_newchunk(), iff_writetechunk(), iff_endchunk(),
 iff_close(), iff_goodtype(), IFFHandle, _SwapBE32()

Author: Ronny Bangsund

40**int iff_newchunk** (IFFHandle* handle, ulong id)*Start on a new chunk in an IFF.*

Start on a new chunk in an IFF. You should finish writing any other chunks first with `iff_endchunk()` before calling this. The supplied chunk identifier will be written in big-endian format.

See Also: `iff_new()`, `iff_writechunkdata()`

Author: Ronny Bangsund

41

```
size_t iff_writetechunkdata (IFFHandle* handle, char* buffer,  
                             size_t size)
```

Write data to an IFF where you have just started a chunk.

Write data to an IFF where you have just started a chunk. Before you can write arbitrary data to an IFF, you must create a chunk with `iff_newchunk()`. You may repeat this call any number of times if not all data is available on the first call.

See Also: `iff_new()`, `iff_newchunk()`
Author: Ronny Bangsund

42

`void list_add (List* list, Node* item)`

Add a Node to a List.

Add a Node to a List. The Node structure passed in is either just a node with space for pointers to previous and next node, and some data, or it can be the head of a larger structure. It will be inserted at the bottom of the List, as element list->tail.

Each time an item is added to a List, its size element is increased by one. This can be used to instantly tell how many items are in the list.

Parameters: `list` A struct List to add item to
 `item` A Node structure to add to list

See Also: `List`, `Node`, `list_new()`, `list_newitem()`,
 `list_insert()`, `list_remove()`, `list_delete()`, `list_free()`,
 `list_nodemakecircular()`

Author: Ronny Bangsund

43

void list_addhead (List* list, Node* item)

Add a Node to the top of a List.

Add a Node to the top of a List. The Node structure passed in is either just a node with space for pointers to previous and next node, and some data, or it can be the head of a larger structure. It will be inserted at the top of the List, as element list->head.

Each time an item is added to a List, its size element is increased by one. This can be used to instantly tell how many items are in the list.

Parameters: list A struct List to add item to.
 item A Node structure to add to list.

See Also: List, Node, list_new(), list_newitem(),
 list_insert(), list_remove(), list_delete(), list_free(),
 list_nodemakecircular()

Author: Ronny Bangsund

44

Node* list_delete (List* list, Node* item)*Remove a Node from a List and free data.*

Remove a Node from a List and free data. If the List has been supplied with a destructor function, this will be called with item as its parameter. Otherwise, a simple call to free() will be made.

Return Value: A pointer to the next Node after item, or NULL.

Parameters: list A struct List to remove item from, and also get a pointer to the destructor
item A Node structure to remove from list

See Also: List, Node, list_new(), list_newitem(), list_add(), list_remove(), list_free()

Author: Ronny Bangsund

45

`void list_free (List* list)`

Free a List and all its attached Node structures.

Free a List and all its attached Node structures. Since list_delete() does the actual freeing of each Node, destructors will be called if available.

Parameters: list A struct List to remove all items from
See Also: List, Node, list_remove(), list_delete()
Author: Ronny Bangsund

46**Node* list_getnode** (List* list, int index)*Return a Node at a specific index position.*

Return a Node at a specific index position. This will simply traverse the List until either a Node is found at position index, or there are no more items in the List.

Return Value: A pointer to a Node, or NULL if the index was too high

Parameters: list A struct List to get a Node from
index Node number to get a pointer to

See Also: List, Node, list_new(), list_newitem(), list_add(), list_insert()

Author: Shane O'Neill

47

void list_insert (List* list, Node* prev, Node* item)

Insert one Node after another given Node.

Insert one Node after another given Node. The Node *item* will be inserted after the Node *prev* in *list*. It will not fail, unless you pass NULL pointers. The size of *list* will also be incremented.

Parameters:

| | |
|-------------------|-----------------------------------|
| <code>list</code> | A struct List to insert Node item |
| <code>prev</code> | Insert item after this |
| <code>item</code> | The Node to be inserted into list |

See Also: `List`, `Node`, `list_new()`, `list_newitem()`, `list_add()`, `list_remove()`, `list_delete()`

Author: Ronny Bangsund

48

void list_makecircular (List* list)

Make a List circular (first and last Node points to eachother).

Make a List circular (first and last Node points to eachother). A circular list is used mostly for efficient looping of animations (which means only a typical, lazy programmer could have thought of it).

Parameters: node A Node used as reference point
See Also: List, Node, list_nodemakecircular()
Author: Shane O'Neill, Ronny Bangsund

49**List* list_new** (void (*destructor)())*Create a new List and set the destructor.*

Create a new List and set the destructor. If no destructor is specified, a simple free() will be called on items in list_delete().

Return Value: If all is well, a pointer to a List
Parameters: destructor A function used to delete each Node. The only parameter is a Node pointer.
See Also: List, Node, list_newitem(), list_add(), list_remove(), list_delete(), list_free()
Author: Ronny Bangsund

50

Node* list_newitem (List* list, void* data)*Allocates a new Node structure.*

Allocates a new Node structure. The node will be initialised with data and inserted into list. If *list* is NULL, it merely creates the new Node and returns a pointer to it.

Return Value: A pointer to a Node, with all three fields initialised

Parameters:

| | |
|------|--|
| list | An optional struct List to insert Node item into |
| data | Data for the data element in the created node. Optional. |

See Also: List, Node, list_new(), list_add(), list_remove(), list_delete()

Author: Ronny Bangsund

51

Node* list_remove (List* list, Node* item)*Remove a Node from a List without freeing any data.*

Remove a Node from a List without freeing any data.

Return Value: A pointer to the next Node after item, or NULL.

Parameters: list A struct List to remove item from
item A Node structure to remove from list

See Also: List, Node, list_new(), list_newitem(), list_add(), list_delete(), list_free()

Author: Ronny Bangsund

52

`void MD5Init (struct MD5Context* ctx)`

Start MD5 accumulation.

Start MD5 accumulation. Set bit count to 0 and buffer to mysterious initialization constants.

To sum a stream of data, you start by calling MD5Init() on the context buffer. Then you pass MD5Update() with context and buffer as your parameters until there is no more data to checksum. Get the final key with a call to MD5Final().

Parameters: ctx The context that all MD5 routines rely on.
See Also: MD5Sum(), MD5Update(), MD5Final(), MD5Ascii()
Author: Ron Rivest (algorithm), Colin Plumb (code)

53

```
void MD5Update (struct MD5Context* ctx, unsigned char
                const* buf, unsigned len)
```

Update context to reflect the concatenation of another buffer full of bytes.

Update context to reflect the concatenation of another buffer full of bytes. Finalise with MD5Final after last part of stream to sum.

Parameters:

| | |
|-----|--|
| ctx | The context, initialised once by MD5Init() |
| buf | A buffer of whatever data you want to sum. |
| len | Size of the buffer. |

See Also: MD5Sum(), MD5Init(), MD5Final(), MD5Ascii()

Author: Ron Rivest (algorithm), Colin Plumb (code)

54

```
void MD5Final (unsigned char digest[16], struct MD5Context*  
               ctx)
```

Final wrapup.

Final wrapup. Pads to a 64-byte boundary with the bit pattern 1 0* (64-bit count of bits processed, MSB-first).

Parameters:

| | |
|--------|---|
| digest | The actual buffer to hold the final key. |
| ctx | The temporary MD5 context to create key from. |

See Also: MD5Sum(), MD5Init(), MD5Update(), MD5Ascii()

Author: Ron Rivest (algorithm), Colin Plumb (code)

55

```
void MD5Sum (unsigned char* key, unsigned char* buf, unsigned len)
```

MD5-checksum a buffer.

MD5-checksum a buffer. If you are not checksumming something piece by piece, this function will do the whole calculation for you. If you need to read in one piece of a stream to sum at a time, you must MD5Init() first, then MD5Update() with each chunk and run MD5Final() when all has been read. Otherwise, MD5Sum() is the tool.

Parameters: key A 16-byte buffer with the MD5 key.
 buf A buffer to sum, adding to the MD5 key.
 len Size of the buffer.

See Also: MD5Init(), MD5Update(), MD5Final(), MD5Ascii()

Author: Ronny Bangsund

56

`void MD5Ascii (unsigned char* key, unsigned char* ascii)`

Make a printable version of the MD5 sum.

Make a printable version of the MD5 sum. This converts a non-printable 16-byte (128-bit) key to ASCII.

| | | |
|--------------------|---|--|
| Parameters: | <code>key</code> | 16-byte buffer with the key after MD5Init(), MD5Update() and MD5Final() are done. |
| | <code>ascii</code> | A buffer of 32 bytes and a NULL terminator, preferably, to hold the printable key. |
| See Also: | <code>MD5Sum()</code> , <code>MD5Init()</code> , <code>MD5Update()</code> , <code>MD5Final()</code> | |
| Author: | Ronny Bangsund | |

57

Node* node_end (Node* node)*Return the last Node in a circular List.*

Return the last Node in a circular List. This function will actually return the last Node in a List, whether it's circular or not.

Parameters: node A Node used as reference point
See Also: List, Node, list_add(), list_remove(), list_delete(),
list_nodemakecircular(), list_nodestart()
Author: Shane O'Neill

58**Node* node_findbycontents** (Node* node, char* nodename)*Return a Node with data partially containing the entire specified string.*

Return a Node with data *partially* containing the entire specified string. This function is useful to find nodes that contain more than just the data you are looking for. Safe to call on a circular chain of nodes.

Parameters: node A Node to start searching at.
 nodename String that must exist in node->data.
See Also: List, Node, preferences, variable, statuscleaner.c
Author: Ronny Bangsund

59**Node* node_findbyname** (Node* node, char* nodename)*Return a Node with data containing the specified string.*

Return a Node with data containing the specified string. This function is useful to find parts of preferences structures. Safe to call on a circular chain of nodes. NOTE: The search starts at the given node.

Parameters: node A Node to start searching at.
 nodename Name of node you are looking for.

See Also: List, Node, preferences, variable

Author: Ronny Bangsund, Shane O'Neill

60

int node_makecircular (Node* node)

Make a chain of nodes circular (first and last Node points to eachother).

Make a chain of nodes circular (first and last Node points to eachother). A circular list is used mostly for efficient looping of animations (which means only a typical, lazy programmer could have thought of it). This function does not necessarily require a List structure.

Do NOT call on a Node in an already circular chain. That's a waste of time ;)

Parameters: node A Node used as reference point.

See Also: List, Node, list_add(), list_remove(), list_delete(),
list_nodeend(), list_nodestart()

Author: Shane O'Neill

61**Node* node_start** (Node* node)*Return the first Node in a circular List.*

Return the first Node in a circular List. This function will actually return the first Node in a List, whether it's circular or not.

Parameters: node A Node used as reference point
See Also: List, Node, list_add(), list_remove(), list_delete(),
list_nodemakecircular(), list_nodeend()
Author: Shane O'Neill

62

`char* strfind (char* haystack, char* needle, int sense)`

Looks for a string within another string.

Looks for a string within another string. This is both a case-insensitive version of, and a wrapper around, `strstr()`. If `sense` is non-zero, `strstr()` will be called. If zero, an insensitive search will be used.

| | | | | | | | |
|----------------------|---|----------|-------------------------|--------|------------------------------------|-------|----------------------------------|
| Return Value: | A pointer to the position in haystack where needle was found, or NULL | | | | | | |
| Parameters: | <table><tbody><tr><td>haystack</td><td>A string to look inside</td></tr><tr><td>needle</td><td>The string to look for in haystack</td></tr><tr><td>sense</td><td>Case-sensitivity (TRUE or FALSE)</td></tr></tbody></table> | haystack | A string to look inside | needle | The string to look for in haystack | sense | Case-sensitivity (TRUE or FALSE) |
| haystack | A string to look inside | | | | | | |
| needle | The string to look for in haystack | | | | | | |
| sense | Case-sensitivity (TRUE or FALSE) | | | | | | |
| Author: | Ronny Bangsund | | | | | | |

63**int strisnum** (char* text)*Check if a string is all numbers and whitespace.*

Check if a string is all numbers and whitespace. Any regular alphabetic symbols or punctuation encountered invalidate the string as a numeric string. This can be used to check a multiple-value string. The string must be NULL-terminated, or the call never ends.

Return Value: TRUE if the string only contains whitespace and numbers.

Parameters: text A string to check.

Author: Ronny Bangsund

64

char* strlower (char* text)*Turn a string into all-lowercase characters.*

Turn a string into all-lowercase characters. Uses `tolower()`, so locale will be handled if supported by the local C library.

Return Value: A pointer to the string passed in
Parameters: `text` A string to change the case of
See Also: `strupper()`, `tolower()`
Author: Ronny Bangsund

65

`strtoken* str_tokenise (char* s, char* delim)`

Create a tokeniser object from a string.

Create a tokeniser object from a string. The supplied string is copied, and this copy is modified with NULL bytes in place of its delimiters. This function does a better job than `strtok()` and `strtok_r()`. Thread-safe, as long as the passed string doesn't get deallocated before the `strdup()` in the beginning finishes.

See Also: `str_freetoken()`, `strtoken`, `strtok()`, `strtok_r()`

Author: Ronny Bangsund

66

```
void str_freetoken (strtoken* t)
```

Free a tokeniser object.

Free a tokeniser object.

Parameters: `strtoken` A `strtoken` structure created by `str_tokenise()`.

See Also: `str_tokenise()`, `strtoken`

Author: Ronny Bangsund

67

`char* strupper (char* text)`

Turn a string into all-uppercase characters.

Turn a string into all-uppercase characters. Uses toupper(), so locale will be handled if supported by the local C library.

Return Value: A pointer to the string passed in
Parameters: text A string to change the case of
See Also: strlower(), toupper()
Author: Ronny Bangsund

68**tagitem* tag_alloclist** (ulong numtags)*Allocate a tag array big enough for numtags items.*

Allocate a tag array big enough for numtags items. This function is dusty, and may disappear soon.

Parameters: numtags The number of tags to hold.
See Also: tag_finditem(), tag_freelist(), tag_getdata(), tag_next()
Author: Ronny Bangsund

69**tagitem* tag_finditem** (ulong tag, tagitem** taglist)*Look for a tag identifier in a taglist, and return a pointer to the tagitem.*

Look for a tag identifier in a taglist, and return a pointer to the tagitem.

Return Value: A pointer to a tagitem entry matching the tag parameter, or NULL.

Parameters: tag A tag id (program/library specific)
taglist A pointer to a tagarray's address

See Also: tag_alloclist(), tag_freelist(), tag_getdata(), tag_next()

Author: Ronny Bangsund

70

```
void tag_freelist (tagitem** taglist)
```

Free a tagarray previously created with tag_alloclist().

Free a tagarray previously created with tag_alloclist(). This function is just as dusty as tag_alloclist().

Parameters: taglist A pointer to a tagarray's address
See Also: tag_alloclist(), tag_finditem(), tag_getdata(), tag_next()
Author: Ronny Bangsund

71

```
ulong tag_getdata(ulong tag, ulong defaultval, tagitem**  
taglist)
```

Find a tagitem's data by tag ID.

Find a tagitem's data by tag ID.

| | | |
|----------------------|---|---|
| Return Value: | Data | in the tagitem structure, or defaultval if not found. |
| Parameters: | tag | The program-specific identifier to look for |
| | defaultval | Value to return if a tag is not found |
| | taglist | A pointer to a tagarray's address |
| See Also: | tag_alloclist(), tag_freelist(), tag_finditem(), tag_next() | |
| Author: | Ronny Bangsund | |

72

`tagitem* tag_next (tagitem** taglist)`

Get next tagitem in the array.

Get next tagitem in the array. Use this function to smooth over any uses of TAG_SKIP, TAG_IGNORE and TAG_MORE.

Return Value: A pointer to the next tagitem in the array, or
NULL (=TAG_END).

Parameters: taglist A pointer to a tagarray's address

See Also: tag_alloclist(), tag_freelist(), tag_finditem(),
tag_getdata(), tag_next()

Author: Ronny Bangsund

73 typedef struct **autofile**

Autofile structure.

Names

| | | | | |
|-------|--------|---|---|-----------|
| 73.1 | int | w | <i>Boolean indicating the file is open for writing.</i> | 78 |
| 73.2 | size_t | size | <i>Size of file.</i> | 79 |
| 73.3 | size_t | pos | <i>Current position in an open file. ...</i> | 79 |
| 73.4 | size_t | read | <i>Number of bytes read in last read operation.</i> | 79 |
| 73.5 | char* | name | <i>Name of file.</i> | 80 |
| 73.6 | char* | buf | <i>A buffer used in file_read() and file_write() operations.</i> | 80 |
| 73.7 | size_t | bufsize | <i>Size of buf element.</i> | 80 |
| 73.8 | FILE* | f | <i>The actual FILE pointer.</i> | 81 |
| 73.9 | void | (*readhook) (char* buf, size_t len) | <i>Hook called when reading.</i> | 81 |
| 73.10 | void | (*writehook) (char* buf, size_t len) | <i>Hook called when writing.</i> | 81 |

Autofile structure. Use file_new() to create a structure with a filename. The other file_* operations can be used to read, write and seek in the file.

See Also: file_new(), file_free(), file_getsize(), file_initbuf(), file_load(), file_seek(), file_scan(), file_read(), file_write(), file_readhook(), file_writehook()

Author: Ronny Bangsund

73.1 int **w**

Boolean indicating the file is open for writing.

Boolean indicating the file is open for writing. It will be created if it doesn't exist.

73.2

`size_t size`

Size of file.

Size of file. Writing operations will add the number of bytes written from each call.

73.3

`size_t pos`

Current position in an open file.

Current position in an open file. This usually equals size with a file being written to.

73.4

`size_t read`

Number of bytes read in last read operation.

Number of bytes read in last read operation. The pos element will point to the point after last byte read. Start point of the last read chunk of data is pos - read.

73.5**char* name***Name of file.*

Name of file. No special checking is made on this name to ensure it is valid for the OS to create.

73.6**char* buf***A buffer used in file_read() and file_write() operations.*

A buffer used in file_read() and file_write() operations. You must create this buffer with file_initbuf(). The buffersize shouldn't need to be larger than the size of a file. You may not always know how large a file being written will be, so a smaller buffer of a few kilobytes is advisable.

73.7**size_t bufsize***Size of buf element.*

Size of buf element. file_read() will not read more than this much per call. The previous buffer contents are of course lost. No special care is made to clear the contents before reading, so do not expect everything to be nicely NULL-terminated.

73.8**FILE* f***The actual FILE pointer.*

The actual FILE pointer. For internal use.

73.9**void (*readhook) (char* buf, size_t len)***Hook called when reading.*

Hook called when reading. This is called right after reading in all data and ensuring anything was read. See md5.c in the base Xiqua directory for an example of simple usage of a hook.

73.10**void (*writehook) (char* buf, size_t len)***Hook called when writing.*

Hook called when writing. Before actually writing, this is called. Note that file_write() may not always be able to write the entire requested buffer (if disk is full, permissions are wrong etc.), so you should compare the return value with the requested size.

74

```
#define xi_textwidth (font, text)
```

Calculate how many pixels wide a string printed with a fixed-width xfont will be.

Calculate how many pixels wide a string printed with a fixed-width xfont will be. This is merely a wrapper for a multiplication of the length of the text and the font width. Use `SDL_ttf` for nice variable width fonts.

Return Value: An integer indicating how many pixels wide the text will be on screen.

Parameters: `font` A font in Xiqua's own, peculiar format.
`text` A string you want to find space for.

See Also: `xifont`, `xi_loadfont()`, `xi_freefont()`, `xi_puttext()`

75

```
#define MAKE_ID (a,b,c,d)
```

Quick macro to make a ulong of four characters.

Quick macro to make a ulong of four characters. This only creates them in internal order. You still need to convert to big-endian format before writing it to an IFF.

The first character is the leftmost in a string; little-endian machines will store this backwards, so `_SwapBE32()` will be required.

Parameters:

- a First character.
- b Second character.
- c Third character.
- d Fourth character.

76

typedef struct **IFFHandle***Handle returned by iff_new() and iff_open() calls.***Names**

| | | | | |
|------|-------|----------------------|--|----|
| 76.1 | FILE* | iff | | 84 |
| 76.2 | ulong | type | <i>FORM type.</i> | 84 |
| 76.3 | ulong | id | <i>Identifier, somewhat less strict.</i> ... | 85 |
| 76.4 | ulong | chunksize | <i>Size of current chunk.</i> | 85 |
| 76.5 | ulong | prevchunksize | <i>Makes the reader able to scan back-wards.</i> | 85 |
| 76.6 | ulong | size | <i>Size of the IFF</i> | 86 |
| 76.7 | ulong | pos | <i>Position in file.</i> | 86 |
| 76.8 | List | chunks | <i>Linked list of chunk context nodes.</i> . | 86 |
| 76.9 | char | write | <i>Are we in write mode?</i> | 86 |

Handle returned by iff_new() and iff_open() calls. All reading/writing functions need this to see where they are, and to store temporary nodes while building an IFF.

See Also: iff_new(), iff_open()

Author: Ronny Bangsund

76.1

FILE* **iff**

76.2

ulong **type***FORM type.*

FORM type. See the IFF85 text for a thorough introduction.

76.3

| |
|-----------------|
| ulong id |
|-----------------|

Identifier, somewhat less strict.

Identifier, somewhat less strict. Used by the reading functions. Soon to be replaced.

76.4

| |
|------------------------|
| ulong chunksize |
|------------------------|

Size of current chunk.

Size of current chunk. Used by the reading functions. Soon to be replaced.

76.5

| |
|----------------------------|
| ulong prevchunksize |
|----------------------------|

Makes the reader able to scan backwards.

Makes the reader able to scan backwards. Used by the reading functions. Soon to be replaced.

76.6**ulong size***Size of the IFF*

Size of the IFF

76.7**ulong pos***Position in file.*

Position in file. Usually the result of an ftell().

76.8**List chunks***Linked list of chunk context nodes.*

Linked list of chunk context nodes. Not used by the reader functions yet. The API is more or less locked down, but behind the scenes, there is some inconsistency between reading and writing of IFF. Fixing it soon.

76.9**char write***Are we in write mode?*

Are we in write mode? iff_close() checks this to see if the final filesize needs to be corrected, and if chunks are to be finalised.

77

```
typedef struct Node
```

A simple Node.

A simple Node. Every programmer probably knows about lists and nodes, so I won't elaborate.

Xigual List functions make no assumptions about the contents and size of a Node structure passed to them. If you have larger structures with data to be freed, the destructor callback is useful.

Author: Ronny Bangsund

77.1

```
struct Node *next
```

Next node in List.

Next node in List. It can be any structure starting with a Node structure, as Xigual does not assume anything about data beyond the Node header.

77.2

```
struct Node *prev
```

Previous Node in List.

Previous Node in List. It can be any structure starting with a Node structure, as Xigual does not assume anything about data beyond the Node header.

77.3

void* data

Node-specific data.

Node-specific data. Xiqua routines use this pointer as a string pointer to a name, or a pointer to sound/graphics data.

78
typedef struct List

*Linked list structure.***Names**

| | | | | |
|------|-------|-------------------------------|---|----|
| 78.1 | Node* | head | <i>First Node in List</i> | 89 |
| 78.2 | Node* | tail | <i>Last Node in List</i> | 90 |
| 78.3 | ulong | size | <i>Number of items in List</i> | 90 |
| 78.4 | void | (*destructor) (void*) | <i>A destructor callback for items (optional)</i> | 90 |

Linked list structure. The first Node will not have a previous Node pointer, and the last will have no next Node pointer. Use the `list_makecircular()` function to make it loop.

See Also: Node, `list_new()`, `list_add()`, `list_insert()`
Author: Ronny Bangsund

78.1
Node* head

First Node in List

First Node in List

78.2
Node* tail

Last Node in List

Last Node in List

78.3

ulong **size**

Number of items in List

Number of items in List

78.4

void (***destructor**) (void*)

A destructor callback for items (optional)

A destructor callback for items (optional)

79

```
#define list_nodemakecircular (n)
```

Wrapper for node_makecircular().

Wrapper for node_makecircular().

80

```
#define list_nodestart (n)
```

Wrapper for node_start().

Wrapper for node_start().

81**#define list_nodeend (n)***Wrapper for node_end().*

Wrapper for node_end().

82

`struct MD5Context`

MD5 context for the Rivest/Plumb MD5 checksumming routines.

MD5 context for the Rivest/Plumb MD5 checksumming routines.

Author: Ron Rivest (algorithm), Colin Plumb (code)

83

typedef struct **preferences***A preferences section.***Names**

| | | | | |
|------|-------|-------------------|--|----|
| 83.1 | char* | name | <i>Name of section</i> | 95 |
| 83.2 | char* | dir | <i>Directory we prefer to save this in</i> . | 95 |
| 83.3 | List | variables | <i>A chain of variables</i> | 96 |
| 83.4 | char | stringonly | <i>Boolean - true if we don't want to convert numbers to integers.</i> | 96 |

A preferences section. Each preferences file may consist of several sections like this.

See Also: `cfg_loadprefs()`, `cfg_newsection()`, `cfg_newvar()`,
`cfg_addvar()`, variable

Author: Ronny Bangsund

83.1

char* **name***Name of section*

Name of section

83.2

char* **dir***Directory we prefer to save this in*

Directory we prefer to save this in

83.3**List variables***A chain of variables*

A chain of variables

83.4**char stringonly***Boolean - true if we don't want to convert numbers to integers.*

Boolean - true if we don't want to convert numbers to integers. When true, the function that scans each line will just blindly copy the variable data to the string element. The type will be set to PREFS_STRING.

84

typedef struct **variable***A variable.***Names**

| | | | | |
|------|-----------------|---------------|---|----|
| 84.1 | struct variable | *next | <i>Next variable in List</i> | 97 |
| 84.2 | struct variable | *prev | <i>Previous variable in List</i> | 98 |
| 84.3 | char* | name | <i>The mandatory name of the variable.</i> | 98 |
| 84.4 | ulong | type | <i>Type of variable, as listed in prefsh</i> | 98 |
| 84.5 | char* | string | <i>A string, if type is PREFS_STRING</i> | 98 |
| 84.6 | int | value | <i>An integer (signed) if type is PREFS_VALUE or PREFS_BOOLEAN.</i> | 99 |

A variable. This extended node contains the type of variable, currently one of **PREFS_STRING**, **PREFS_VALUE** and **PREFS_BOOL** and the related data.

See Also: preferences, cfg_newvar()

Author: Ronny Bangsund

84.1

struct variable ***next***Next variable in List*

Next variable in List

84.2

`struct variable *prev`

Previous variable in List

Previous variable in List

84.3

`char* name`

The mandatory name of the variable.

The mandatory name of the variable. `cfg_*` functions will search by this only. Anything recognised by `isalpha()` is legal as a variable name. Locale settings may affect what is considered an alphabetical character.

84.4

`ulong type`

Type of variable, as listed in prefsh

Type of variable, as listed in prefsh

84.5

`char* string`

*A string, if type is **PREFS.STRING***

A string, if type is **PREFS.STRING**

84.6**int value**

*An integer (signed) if type is **PREFS_VALUE** or **PREFS_BOOLEAN**.*

An integer (signed) if type is **PREFS_VALUE** or **PREFS_BOOLEAN**. If a lone word without an equals sign is encountered in a configuration file, it is turned into a boolean variable.

85 typedef struct **xiview**

This is a view, for use as sprites, message boxes, or anything else that might need to store the rectangle it is blitted over.

Names

| | | | |
|-------|---------------|---------------|---|
| 85.1 | struct xiview | *next | <i>Next view, if used in a List</i> 101 |
| 85.2 | struct xiview | *prev | <i>Previous view, if used in a List</i> 101 |
| 85.3 | char* | name | <i>Optional name for this view</i> 101 |
| 85.4 | SDL_Surface* | store | <i>Rectangle we blitted this view over (available by default)</i> 101 |
| 85.5 | SDL_Surface* | image | <i>The image to blit onto screen</i> 102 |
| 85.6 | SDL_Surface* | screen | <i>This is the SDL_Surface that this view is tied to.</i> 102 |
| 85.7 | ulong | flags | <i>Various xiview flags, as documented in view_new()</i> 102 |
| 85.8 | Uint32 | bghcol | <i>This will be used as fill colour if XIV_FILLBACK is defined in flags, or a colourkey unless XIV_NOKEY is defined</i> 102 |
| 85.9 | int | w | <i>Width of area to blit onto</i> 103 |
| 85.10 | int | h | <i>Height of area to blit onto</i> 103 |
| 85.11 | int | x | <i>X position in screen</i> 103 |
| 85.12 | int | y | <i>Y position in screen</i> 103 |
| 85.13 | char | hidden | <i>TRUE if hidden by view_hide()</i> 104 |

This is a view, for use as sprites, message boxes, or anything else that might need to store the rectangle it is blitted over.

See Also: `view_new()`, `view_show()`, `view_hide()`
Author: Ronny Bangsund

85.1

`struct xiview *next`

Next view, if used in a List

Next view, if used in a List

85.2

`struct xiview *prev`

Previous view, if used in a List

Previous view, if used in a List

85.3

`char* name`

Optional name for this view

Optional name for this view

85.4

`SDL_Surface* store`

Rectangle we blitted this view over (available by default)

Rectangle we blitted this view over (available by default)

85.5**SDL_Surface* image***The image to blit onto screen*

The image to blit onto screen

85.6**SDL_Surface* screen***This is the SDL_Surface that this view is tied to.*

This is the SDL_Surface that this view is tied to. Note that there is nothing stopping you from using any old surface to blit onto.

85.7**ulong flags***Various xiview flags, as documented in view_new()*

Various xiview flags, as documented in view_new()

85.8**Uint32 bgscol***This will be used as fill colour if XIV_FILLBACK is defined in flags, or a colourkey unless XIV_NOKEY is defined*

This will be used as fill colour if XIV_FILLBACK is defined in flags, or a colourkey unless XIV_NOKEY is defined

85.9

int **w**

Width of area to blit onto

Width of area to blit onto

85.10

int **h**

Height of area to blit onto

Height of area to blit onto

85.11

int **x**

X position in screen

X position in screen

85.12

int **y**

Y position in screen

Y position in screen

85.13**char hidden***TRUE if hidden by view_hide()*

TRUE if hidden by view_hide()

Xigual questions and answers.

<h1>Xigual</h1>

1. What is it? A collection of libraries that help me writing SDL programs, and also do many other things.
2. What's in it? Several sub-libraries:
 - (a) xitools This is the core library. All the other sub-libraries require this to some extent, particularly for linked lists and all the related manipulation, but sometimes also for file-handling.
 If you need to read some configuration files, the `cfg_*` functions will do most of what you need. Hooks are supported to do your own processing of each variable, although this bit needs some more documentation.
 The `iff_*` functions help you create EA-IFF85 style files. It's a simple standard for binary data, with some requirements that make them useful for portable data. The basic FORM types can be used for pretty much any purpose. You may know these filetypes from the Amiga platform.
 Taglists also have some utility functions. If you want to avoid changing the interface every time you feel a new option should be supported, this is the way to do it. Using `varargs`, you can make your API permanent, only adding more tags for new features. Highly recommended for plugin interfaces and dynamic libraries. If only the Gtk+ developers used tags..
 MD5 calculation has been grabbed off the 'net, with one additional function by me to create a printable string. Probably the most portable function in the entire library :/
 The string tokeniser in Linux' C library, `strtok()`, is pretty daft. Even the more recent `strtok_r()` isn't very good, as it's not very portable. They both have many bugs, so I had to write a new one. Use `str_tokenise()` to create a list of tokens more portably. Unlike `strtok()`, it should be thread-safe.
 - (b) xigfx You'll find the initialisation functions here. `xi_init()`, `xi_main()` and `xi_quit()` are the important calls. Utility functions for graphics are here, too - views and tilesets are the only really evolved ones included. A map loader, displayer and scroller is in the works, using tilesets as a foundation.
 - (c) xicgi I started writing some utility functions for CGI creation, and couldn't find a better place to put them. If there are other pervs out there who like to write everything in C, this could be useful. Cookies are supported, too.
 - (d) xinet This sub-library only has a server-toolkit for now. It can create select-servers (non-threaded servers). Mostly untested; works well locally, but needs larger scale testing.

- (e) xithings Various functions to handle objects, things, thingamajigs. You know, players, monsters, inventory objects, magic, whatever. Scripting support will somehow be integrated into this later.
3. What's needed to use and compile it? Development is happening under Linux, as is fashionable these days. Several of the libraries have compiled nicely on many common Unix- like platforms, even without GNU cc. Unless there is a MacOS X port of SDL, that probably won't work, but the other parts worked on the SourceForge compile farm.
- Recommended setup:
 Unix-derivative OS with SDL (<http://libsdl.org>), SDL_image, SDL_mixer and SDL_ttf, at least the GNU C compiler and proper headers, plus GDB. DDD is a nice frontend, and Valgrind has helped me squash many memory bugs. A profiler would be a nice addition, but you'll be distracted by the lack of optimisation in Xiqua itself.
4. What about Win32? Ah..yes..Windows..I am *almost* able to compile Xiqua with my cross-compiler now, but very far away from getting it to work with VC++ 5.x. SDL is meant to ease cross-platform development, but I'm afraid much of my non-SDL code is tied to Unix and POSIX code. Getting a GCC-derivative to compile Xiqua natively or via cross- compilation is on the list of things I want. Some of my projects are just too unique to leave anyone out ;)
- I've been able to make MingW32 compile Xiqua with only a few warnings, but linking with test-programs has not been successful. In some cases, I am sure MingW32 simply lacks certain POSIX/BSD/SysV functions in its C library, but other times I suspect my code is the problem. Any help is appreciated, and gives you a position in the AUTHORS file.
5. Right..and C++? Much better support there. No Xiqua wrapper class has been written, but it should link. C links with nearly anything. The headers have definitions to ensure C++ compatibility, thanks to Mr. O'Neill.
6. What's the license? LGPL. See the file of the same name. If you find any code of use to you, feel free to simply use the sourcefile and related headers. Just abide by the LICENSE and all is well. If you think I'm doing anything uncool, let me know.
7. You mentioned SDL..what the heck is that? Simple Directmedia Library. The name gives some indication what it is..basically, it's a canvas for portable graphics programming. It supplies the basic functions to create a 2D-display (and also OpenGL with recent versions) and handle input from keyboard, mouse and other controllers, timers, sound and threads (on some platforms), plus even CD-ROM access. All this works in Win32, Linux and MacOS, plus a lot of Unix-derivatives.
- SDL has been used to port many games to Linux, and I'd recommend it for input handling with OpenGL. For 2D programming, it has blitting functions, image loading (extended a lot with SDL_image) and all sorts of low-level screen manipulation. But it is merely a canvas; you need to supply the drawing tools. Read up on Bresenham's algorithm ;)
- Get SDL from <http://libsdl.org>, and grab SDL_image, SDL_mixer and SDL_ttf while you're at it.

8. What does the name mean? It's Ouranian-Barbaric for "manifest". It's a verb.

9. How do I pronounce it? How would you like to pronounce it?

<h6> Clumsily written in one of the headers.. Note the funny way DOC++ mangles this document ;) </h6>

Author: Ronny Bangsund, Shane O'Neill

87
 typedef struct **xiglob**

Global structure for Xigual's automatic handling of miscellaneous data

Names

| | | | | |
|-------|-----------------|---|---|------------|
| 87.1 | SDL_Surface* | screen | <i>This is the SDL surface returned by SDL_SetVideoMode()</i> | <i>109</i> |
| 87.2 | xiview* | cursor | <i>An xiview for the main cursor</i> | <i>109</i> |
| 87.3 | int | hotx | <i>Cursor X offset</i> | <i>109</i> |
| 87.4 | int | hoty | <i>Cursor Y offset</i> | <i>109</i> |
| 87.5 | SDL_Surface* | tscreen | <i>Default surface for tile blits.</i> | <i>110</i> |
| 87.6 | struct xitiles* | tileset | <i>Default tileset to blit from</i> | <i>110</i> |
| 87.7 | int | audio_rate | <i>Sample rate.</i> | <i>110</i> |
| 87.8 | int | audio_chan | <i>Number of channels to use.</i> | <i>110</i> |
| 87.9 | int | numchans | <i>Maximum number of channels to mix.</i> | <i>111</i> |
| 87.10 | int | audio_bufsize | <i>Size of audio buffers.</i> | <i>111</i> |
| 87.11 | Uint16 | audio_fmt | <i>Audio format.</i> | <i>111</i> |
| 87.12 | List | sounds | <i>Sound nodes.</i> | <i>111</i> |
| 87.13 | List | music | <i>Music nodes.</i> | <i>112</i> |
| 87.14 | List | images | <i>Image nodes.</i> | <i>112</i> |
| 87.15 | List | sprites | <i>Sprite nodes.</i> | <i>112</i> |
| 87.16 | void | (*mainloop) (Uint8* keystate, SDLMod modstate, Uint8 mousestate, int mx, int my) <i>User-definable hook, called once each time xi_main() is called.</i> | | <i>113</i> |

Global structure for Xigual's automatic handling of miscellaneous data

87.1
 SDL_Surface* **screen**

This is the SDL surface returned by SDL_SetVideoMode()

This is the SDL surface returned by `SDL_SetVideoMode()`

87.2

`xiview* cursor`

An xiview for the main cursor

An xiview for the main cursor

87.3

`int hotx`

Cursor X offset

Cursor X offset

87.4

`int hoty`

Cursor Y offset

Cursor Y offset

87.5

SDL_Surface* tscreen*Default surface for tile blits.*

Default surface for tile blits.

See Also: `xit_setdefaults()`, `xit_qdraw()`

87.6

struct xitiles *tileset*Default tileset to blit from*

Default tileset to blit from

87.7

int audio_rate*Sample rate.*

Sample rate. Measured in samples per second. Default is 44100 Hz.

87.8

int audio_chan*Number of channels to use.*

Number of channels to use. Use 1 for mono and 2 for stereo.

87.9**int numchans***Maximum number of channels to mix.*

Maximum number of channels to mix. Default is eight.

87.10**int audio_bufsize***Size of audio buffers.*

Size of audio buffers. This doesn't have to be a particularly large number, but must be a factor of two. Defaults to 4096 bytes.

87.11**Uint16 audio_fmt***Audio format.*

Audio format. The format definitions are in `SDL_mixer.h`.

87.12**List sounds***Sound nodes.*

Sound nodes. Contents TBA.

87.13**List music***Music nodes.*

Music nodes. Contents TBA.

87.14**List images***Image nodes.*

Image nodes. Contents TBA.

87.15**List sprites***Sprite nodes.*

Sprite nodes. Contents TBA.

87.16**void (*mainloop)** (Uint8* keystate, SDLMod modstate, Uint8
mousestate, int mx, int my)*User-definable hook, called once each time xi_main() is called.*

User-definable hook, called once each time `xi_main()` is called.

| | | |
|--------------------|---|--|
| Parameters: | <code>keystate</code> | A pointer to an array of <code>Uint8</code> . Currently pressed keys are non-zero. |
| | <code>modstate</code> | State of modifier keys |
| | <code>mousestate</code> | Flags for current mouse state (buttons). |
| | <code>mx</code> | Current X position of mouse cursor within the main SDL surface |
| | <code>my</code> | Current Y position of mouse cursor within the main SDL surface |
| See Also: | <code>xi_main()</code> , <code>SDL_input.h</code> | |
| Author: | Ronny Bangsund | |

88

typedef struct **strtoken***The token structure.***Names**

| | | | | |
|------|--------|---------------|---|------------|
| 88.1 | char* | s | <i>A copy of the original string.</i> | 114 |
| 88.2 | size_t | len | <i>The length of the string memory. ..</i> | 114 |
| 88.3 | char* | d | <i>The delimiter string</i> | 115 |
| 88.4 | size_t | dlen | <i>Length of the delimiter string.</i> | 115 |
| 88.5 | List | tokens | <i>The individual tokens or words. ...</i> | 115 |

The token structure.

See Also: `strtokenise()`, List

88.1

char* **s***A copy of the original string.*

A copy of the original string. There will be "holes" where the delimiters were, so use the tokens list to access.

88.2

size_t **len***The length of the string memory.*

The length of the string memory. Used internally, but might be useful to users.

88.3**char* d***The delimiter string*

The delimiter string

88.4**size_t dlen***Length of the delimiter string.*

Length of the delimiter string. Maybe not the most useful element.

88.5**List tokens***The individual tokens or words.*

The individual tokens or words. The size element in the list will contain the number of tokens. Each token is a plain Node structure with its data pointer being a pointer to a string (the token).

89typedef struct **tagitem***A tagitem.***Names**

| | | | | |
|------|-------|-------------|--|------------|
| 89.1 | ulong | tag | <i>A program/library-specific ID.</i> | 116 |
| 89.2 | ulong | data | <i>The tagitem's data.</i> | 116 |

A tagitem. Many Xiqua function calls accept an array of these as its sole parameter.

See Also: tag_alloclist(), tag_freelist(), tag_finditem(),
 tag_getdata(), tag_next()

Author: Ronny Bangsund

89.1ulong **tag***A program/library-specific ID.*

A program/library-specific ID. Should be fairly unique within the program or library, and not equal to one of TAG_DONE, TAG_IGNORE, TAG_MORE or TAG_SKIP.

89.2ulong **data***The tagitem's data.*

The tagitem's data. Can be just about anything that fits in a ulong. Common uses are flags, pointers to further data/strings or simply integers.

90

#define XI_VARARG (call, ret)*Macro for vararg taghandling.*

Macro for vararg taghandling. This macro is used internally in Xigual to process the taglists passed to many of its functions. It may also be of use if anyone else wants to make use of tags in their functions.

Examples of its use can be found in the Xigual sources. Basically, the calling method in Xigual has the function taking arguments (ulong tag1, ...), then calling XI_VARARG(function_nameA((tagitem *)&tag1), argtype). Not much easier, but less typing.

| | |
|----------------------|--|
| Return Value: | Whatever you pass as parameter ret. |
| Parameters: | <p>call A function call, passing (tagitem *)&tag1 as one of its parameters. You take all the parameters not automatically handled by this macro in this.</p> <p>ret A typedef for the returnvalue. Note: this must be a pointer. Not very flexible, but these macros are for special purposes, anyway.</p> |
| See Also: | XI_VOIDARG(), va_start(), va_arg(), va_end() |

91

`#define XI_VOIDARG (call)`

Macro for vararg taghandling.

Macro for vararg taghandling. This works like XI_VARARG(), but has no return value.

Parameters: call As XI_VARARG().

See Also: XI_VARARG() , va_start(), va_arg(), va_end()

92

typedef struct xitle*A tileset in-memory representation.*

A tileset in-memory representation. This is the structure given to each tileset when loaded from an IFF. See the tileset program for a description of the file formats used.

See Also: `tileset`, `xitle.loadtiles()`, `xitle.draw()`, `xitle.qdraw()`

93

`#define xit_setdefaults (screen, tiles)`

Set default screen and tileset

Set default screen and tileset This call just puts the supplied pointers into the appropriate places in the global structure. It could be a macro, probably, but shouldn't be necessary to call more than once in the lifetime of a program.

Parameters: `screen` An `SDL_Surface` to blit to by default.
 `tiles` Tileset to use for quick blits.

See Also: `xit_loadtiles()`, `xit_qdraw()`, `tileset`

Author: Ronny Bangsund