Game description

Game is played on a grid of 15x15. Players start the game from each corner of the board and the goal is to control the flag in the middle of the game board. The player who has the control of the flag scores points on each second the player holds the flag. Scoring flag is the size of one grid cell which means that only one unit can occupy the flag at a time. Game uses ascii graphics where players are represented in different colours and units in different icons.

Game can be played with 2-4 players. Each player has two soldier units and two tank units at the beginning of the round. Game round lasts two minutes or until only one player is left alive. If game ends because round time elapsed the winner is the player who has the most points. Otherwise winner is the sole surviving player

Players select their units using number keys from 1-4. After the unit is selected the unit can be set to move vertically, horizontally or diagonally by selecting the direction from numpad keys. Unit will continue movement to assigned direction until it hits a board boundary, other unit or the flag. Player can stop unit movement by pressing key 5 on numpad.

Units attack automatically to the enemies in adjacent grid cells without an explicit command from a player. Each unit deals damage to all adjacent cell locations where enemy units are located.

Game design

Used language is C and user interface will be implemented using ncurses. Game communication will be implemented using UDP. Chat is implemented with TCP.

Gameplay

Joining the game

- After player starts the client program the client will get list of available game servers from the main server.
- Game server list is shown to the player. Player sees the game servers and ping time tho that server. Now player selects the game server.
- Client joins to the game server in where he is assigned to a game, which is not yet started.
- Client must then wait other players to join. After two players have joined the game the other two players have 30 seconds to join or game will start as a two-player game.

Playing the game

- Players start from their each corner. Each player goal is to occupy the flag grid in the middle of the screen in order to score points. Points are scored based on the time unit spends on top of the flag.
- Players get 2 soldier and 2 tank units.
- Unit is selected by using numbers 1-4.
- Unit move direction is set from numpad. Direction can be horizontal, vertical or diagonal. Movement continues until stopped from num 5 or if unit collides with another unit, flag or board boundary.
- Unit deals damage to all adjacent grid locations.

Winning the game

- Game ends when round time of 2 minutes is reached.
- Or if only one player is left.
- Winner is the player who has most points or the sole surviving player.
- Winner is announced to all players.
- Players are returned to the main server.

Game applications

Game consists of three main applications: main server, game server and client.

Main server

- Handles the game service discovery by providing list game server addresses to clients.
- Provides a chat capability between clients in the lobby.
- Main server updates the game server list by polling each game server in once a second. If main server does not get response from game server in 10 consecutive polls, the game server is removed from the list.
- When game server program is started it registers itself to the main server list.
- When game server program is ended it unregisters itself from the main server list.

Game server

- Can run only a single game..
- Provides a chat capability between clients in the game.
- Handles the game state of each running game.
    - Game state is calculated by receiving key press commands from each player.
    - Has the authoritative game state of game board and all game objects.
- Calculate latency to each client by using poll message round trip time.
- Send snapshot of game board and unit state to each client.
    - Snapshot of game board is sent as delta update.

    - If fog of war is enabled, send only those enemy units to player which his units can see.
- Send enemy unit state updates to clients so client can display and predict other player unit movements.
    - If fog of war is enabled, send only those enemy unit states what player units can see.
- Game server frame rate is 20 (50ms).

Client
- The player interface to the game.
- During game server selection the client polls each received game server and measures ping time to each server. This time is shown to the player at the selection screen.
- Handles independent game state simulation.
- When player change unit move state by pressing move or stop button. The client sends the key press command to the game server.
- Occasionally client will receive the game state and sync to the authoritative state.
- Client frame rate is 40 (25ms).

Units

Unit	**Speed**	**Health**	**Damage per hit**	**Grid Icon**
Soldier	180	100	2	S
Tank	80	220	4	T

Unit attribute values can vary between 0-255 By adjusting values during testing the game can be balanced.

User interface mockup

Ncurses interface will be implemented only to the client program. Main server and game server will print the game log to the display.

Game lobby. When client is connected to main server.

Debug and system information, last 5 lines. Can be hidden with a parameter. [log row] [log row] [log row] [log row] [log row]
List of game servers. [server name], [ping], [player list] [server name], [ping], [player list] [server name], [ping], [player list] [server name], [ping], [player list] [server name], [ping], [player list]
Chat, last 5 lines. [sender nick]: [chat message] [sender nick]: [chat message] [sender nick] private: [chat message] [sender nick]: [chat message] [sender nick]: [chat message]
Command: /join server1

d.

Debug and system information, last 5 lines. Can be hidden with a parameter.
[log row]
[log row]
[log row]
[log row]
[log row]

🟦	🟦												🟩 T	🟩 S
🟦	🟦												🟩 S	🟩 T
				T										
		T	S											
				S										
							⬜ F							
									S			T		
										S				
🟨 T	🟨 S								T				🟥	🟥
🟨 S	🟨 T												🟥	🟥

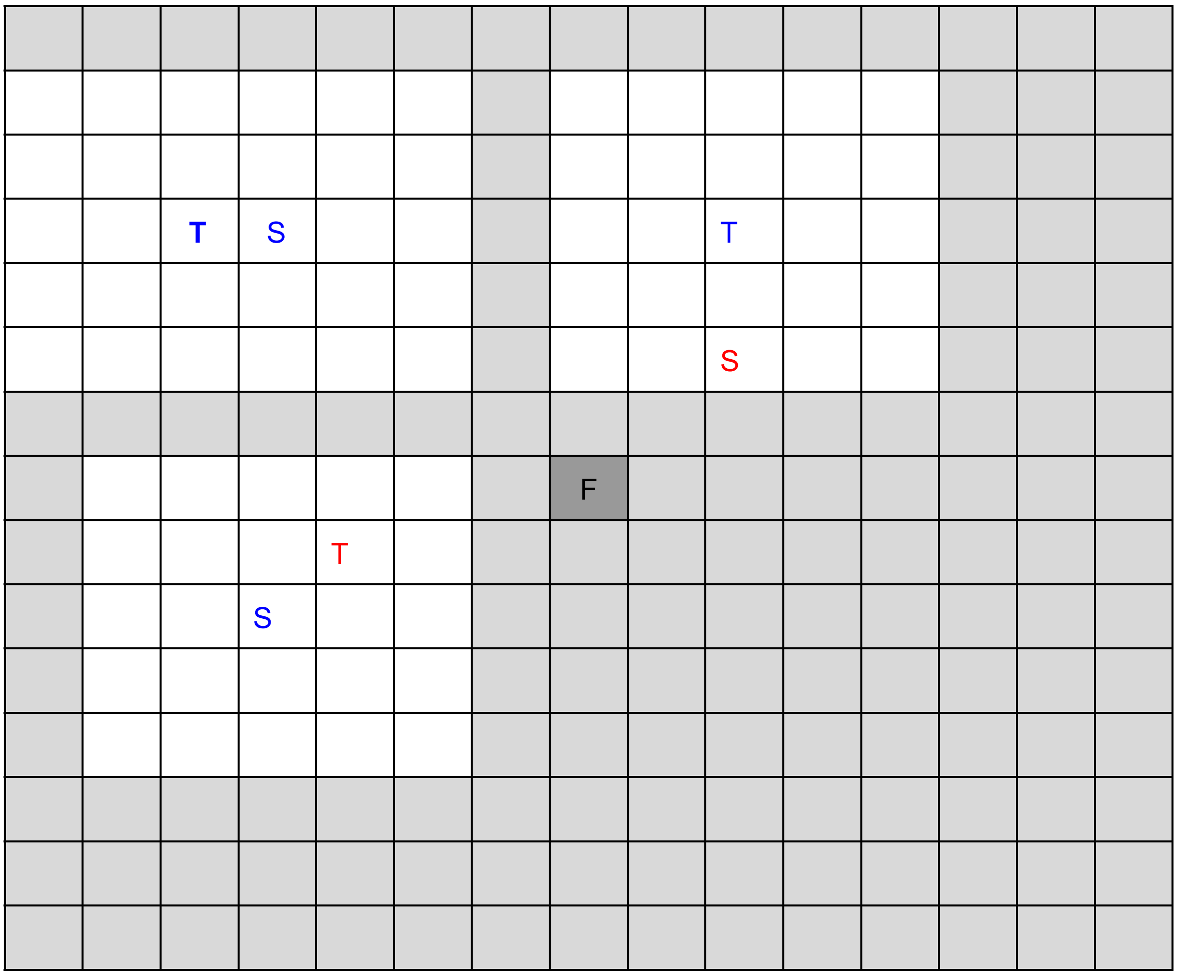
Chat, last 5 lines.
[sender nick]: [chat message]
[sender nick]: [chat message]
[sender nick],private: [chat message]
[sender nick]: [chat message]
[sender nick]: [chat message]

Command: /c nickfoo terve

Game from a blue player perspective. Note that currently selected unit is highlighted.

Gameplay. Fog of war example.

Debug and system information, last 5 lines. Can be hidden with a parameter.
[log row]
[log row]
[log row]
[log row]
[log row]



Chat, last 5 lines.
[sender nick]: [chat message]
[sender nick]: [chat message]
[sender nick],private: [chat message]
[sender nick]: [chat message]

[sender nick]: [chat message]
Command: /c nickfoo terve

Game from a blue player perspective. Note that currently selected unit is highlighted.

Protocol design

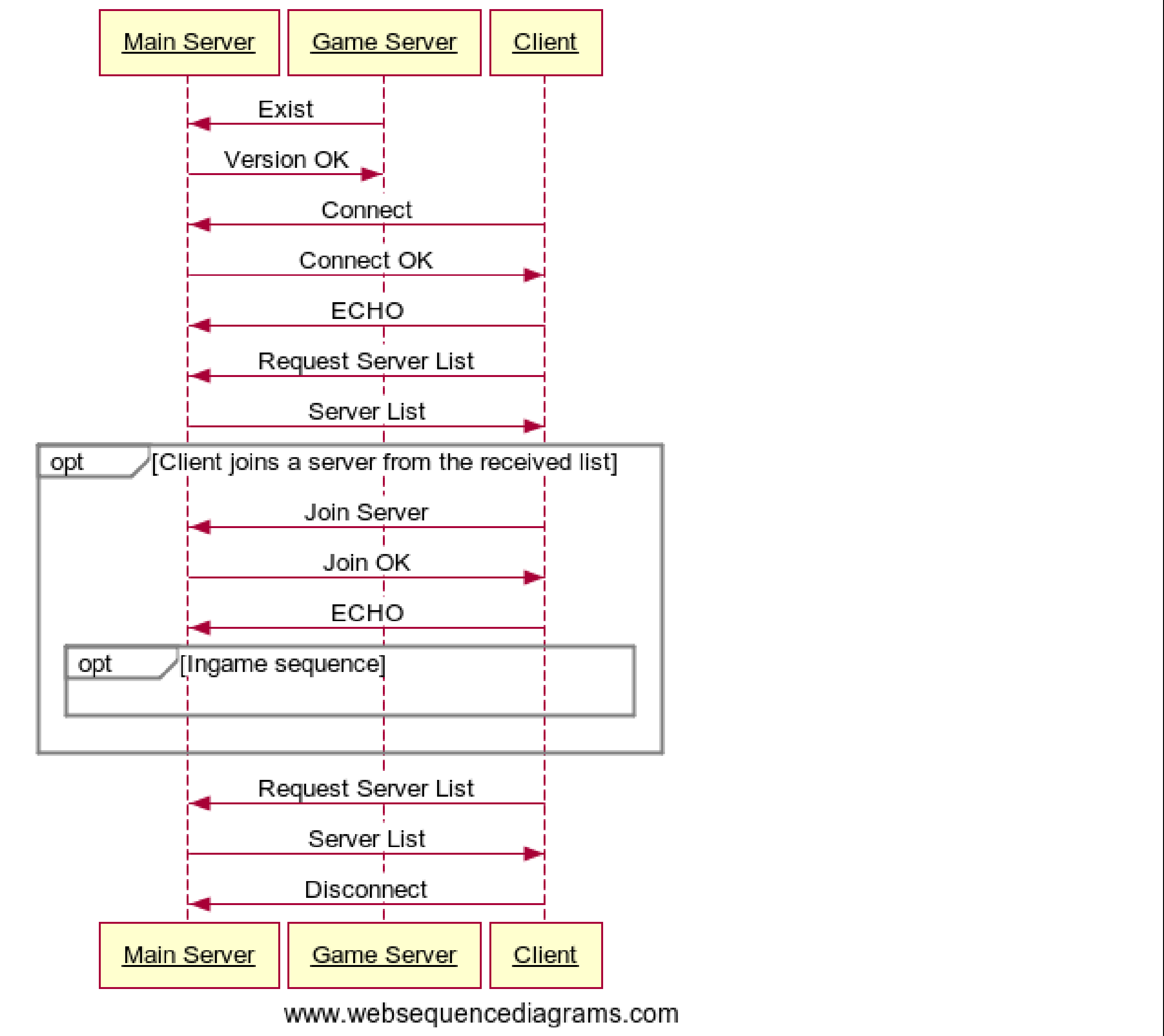
All (UDP) messages start with message type, 8 bit integer. 0-49 are reserved for main server - game server communication, 50-99 for main server - client and >100 for game server - client. Wrong message types are ignored. 255 marks error messages followed by 8 bit integer error code.

Refer to spreadsheet
jUTJSMFNs
RngzRFE&usp=drive_web#gid=0 for UDP packet details.

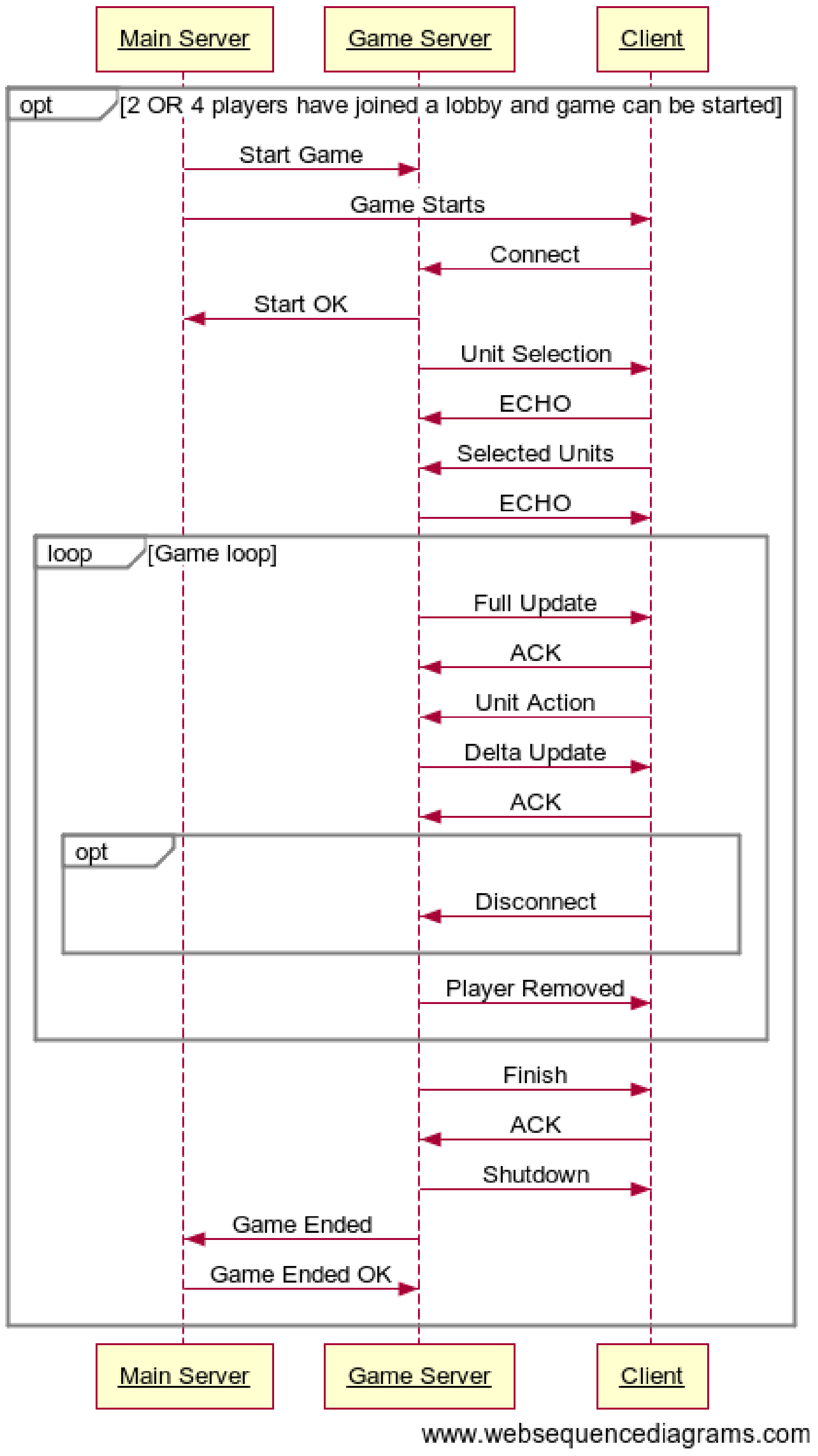
All out-of-game packets are resent with configurable interval until a response is received. Response packets are determined by the communication state, these packets can be unique responses or simple echoes if the state will not necessarily change immediately.

All game state update packets from game server to client are acknowledged by client, depending on the number of the latest acknowledged packet game server will determine whether to send a full update or a delta update to the client.

Error responses are omitted from the diagrams for the sake of clarity. Generally error situations will lead to the termination of the connection.



Out-of-game UDP communication diagram



Ingame UDP communication diagram

TCP Chat

Client sends chat messages to server and server sends the chat message to all client or one specified client. When client is connected to the main server the chat messages are relayed by it. If client is connected to the game server, that server will relay the messages.

Client receives the initial TCP connection parameters through UDP connection.

Chat message from client to server:

8 bit int	32 bit int	32 bit int	n chars	1 byte	n chars	1 byte	n chars	1 byte
Message ID (100)	Total length	Connetion token	Sender nick	\0	Receiver nick or empty if public	\0	Message	\0

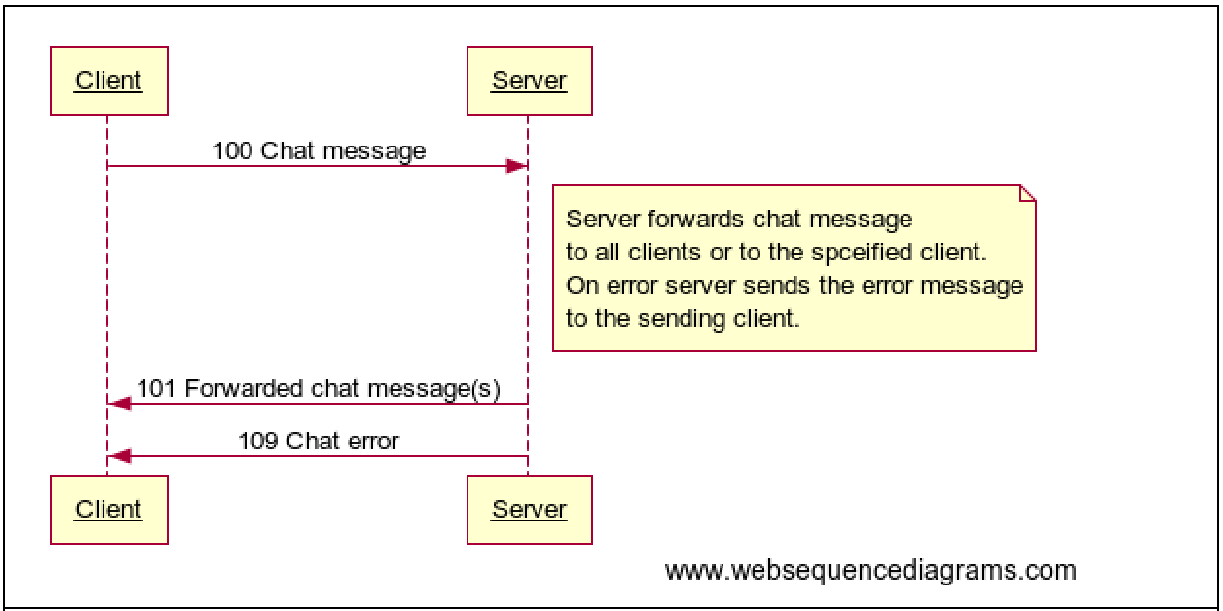
Forwarded chat message from server to all client(s), including the sending client.

8 bit int	32 bit int	n chars	1 byte	n chars	1 byte	n chars	1 byte
Message ID (101)	Total length	Sender nick	\0	Receiver nick or empty if public	\0	Message	\0

Chat error from server to sending client.

8 bit int	32 bit int	8 bit	n chars	1 byte
Message ID (109)	Total length	Error code	Error message	\0

Error code 1: An invalid nick.


Client
Server
100 Chat message
Server forwards chat message to all clients or to the spceified client. On error server sends the error message to the sending client.
101 Forwarded chat message(s)
109 Chat error
Client
Server
www.websequencediagrams.com

Tcp chat sequence.

Planned basic features

All required basic features will be implemented in the game. Game can be played by 2-4 players and there will be winning and ending conditions (flag capture time, destruction of all units and timelimit).

Client server design

Only server runs game simulation. Clients send player moves to the game server and game server sends game world (delta or snapshot) to clients.

Server search

Main server has list of servers and player can choose game server in lobby. Game servers send add/remove from server list packets. Main server also pings game servers and remove those that does not respond after 10 tries.

Chat

Between clients in main server lobby. Both public and private messages will be supported. Uses TCP.

Latency and jitter measurements

Latency displayed in server list in main server lobby. During game, game server measures latency and jitter by sending ping packets to clients.

Testing

Module testing will be done trough whole developing process. Beta version will be tested by 3-4 person group (not us).

Planned advanced techniques

Fog of war

Server sends only location of those units that player units can see.

Packet compression

Packet size is reduced by using deltas - only changed gameworld information is sent to clients.

Real time game.

Game servers runs 20fps speed. Clients at 40fps.

Multi-server system

Multi-server system provides backup server to game server. Game server sends clients information at the beginning of the game to backup server and after that snapshots. If backup server stops receiving packets from game server it starts running game simulation based on last received snapshot. Then it starts sending deltas/snapshots to clients using same token as game server. Clients automatically send their messages to same address from where they receive data and changing server should happen relatively smoothly.

Work schedule

Deadline	**Description**
31.12.2013 23:59	Initial game idea and design document
20.01.2014	Networking code implemented.

30.01.2014	First playable alpha with main server, game server and client with ui.
31.01.2014 12:00	Mid return
05.02.2014	Beta with all basic features.
09.02.2014	Beta version. With all advanced features.
10.02.2014	Game testing.
18.02.2014	Release version of the game.
20.02.2014 10:00	Document return
21.02.2014 09:00	Presentation day
26.02.2014 23:59	Final code return

Coding conventions

- Line change before open bracket.
- Ident lines with tab. (4 space length tab).
- Function input variables end with underline
    - int startUdpServer(char* port_);
- Global variables start with underline.
    - extern int _udpServerSocketFd;
- Comments above functions.
- Pointers in header file belong to type.
    - int startUdpServer(char* port_);
    - int* _port;
- Pointers in code file belong to variable.
    - int startUdpServer(char *port_);
    - int *port;

Division of work

Documentation - Aki, Mikko, Ville

- Game design
- Protocol design
- Code architecture design

Main server

- UDP
    - interface - Aki
    - communication, messages -Mikko
- TCP
    - interface - Aki
    - chat, messages -Aki
- Working logic -Ville

Game server

- UDP
    - interface -Aki
    - communication, messages -Mikko
- TCP
    - interface -Aki
    - chat, messages -Aki
- Game logic -Ville, Mikko, Aki
    - Main loop
    - Message handling

Client

- UDP
    - interface -Aki
    - communication, messages -Mikko
- TCP
    - interface -Aki
    - chat, messages -Aki
- UI (ncurses)
    - Lobby page -Ville
    - Game page -Ville
- Game logic (no game simulation) -Ville, Mikko, Aki
    - User input handling
    - Message handling
    - Command handling (quit, chat)

Testing reports

List of additional libraries

Known issues

References

Description	**Link**