

P.O. Box 1295
NL7500BG Enschede
Tel.: +31 53 4305195
Fax.: +31 53 4329365
<http://www.dgtprojects.com>
Email: info@dgtprojects.com

DGT Electronic Board DLL

Quick guide - Version 1.20

Author: G.H.O. Reitsma
Date: Nov 20th, 2003

© Copyright 2003 DGT Projects B.V. All rights reserved.

1 DGT Electronic Board DLL Description

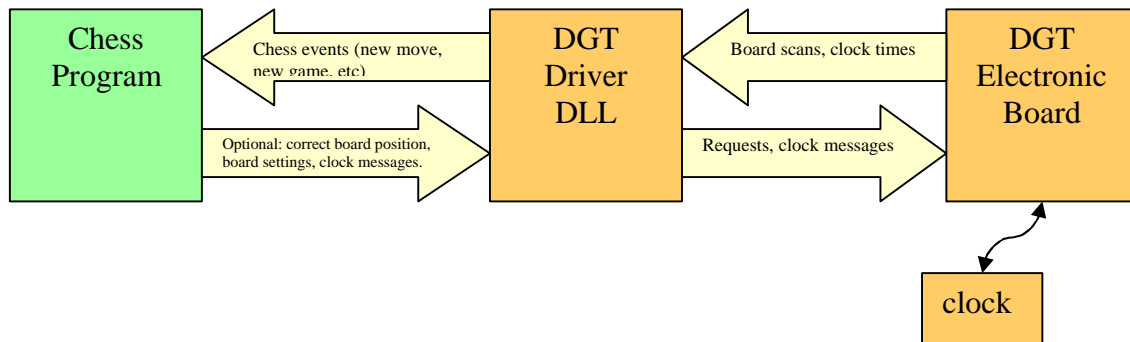
The DGT Electronic Board DLL acts as a layer between the DGT electronic board and the chess program. It reads the chess positions and clock times from the board and converts them to chess events.

For every chess event, the chess program can (optionally) add a pointer from the DLL to a function in the chess program that handles the event. All chess events generated by the DLL are described in chapter 3.

The addition of these pointers to custom event-handling functions is done by “register” functions. For example, the DLL function “_DGTDLL_RegisterScanFunc”. Chapter 5 contains a list of all the functions in the DLL.

Besides these register functions, the DLL has a build-in set-up dialog, where the user can choose the serial port and see the current board position.

Several input functions are described in chapter 4.



2 Board and move representation

2.1 Board representation

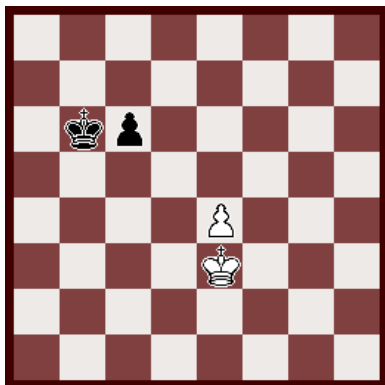


Diagram 1: 17kp17P7K

The board position is represented by a string, based on the Forsyth Edwards Notation (FEN).

A typical board representation string looks like this: “**17kp17P7K**”. (Meaning: 17 empty squares, a black king, a black pawn, 17 empty squares, a white pawn, 7 empty squares, a white king).

This string represents the board position in diagram 1.

2.1.1 Syntax

2.1.1.1 Piece codes

The piece-codes are the same as in FEN:

- **P** for a white pawn, and **p** for a black pawn
- **N** for a white knight, and **n** for a black knight
- **B** for a white bishop, and **b** for a black bishop
- **R** for a white rook, and **r** for a black rook
- **Q** for a white queen, and **q** for a black queen
- **K** for a white king, and **k** for a black king

2.1.1.2 Empty Square:

An empty square is represented by a dot (“.”) or the digit 1. (Meaning: one empty square).

2.1.1.3 General description

A board is described by a single string. Every square is represented by a single character in the string. The first character is the code of the piece on square a8, the next character is the code of the piece on square b8, etc, and the last character of the string is the code of the piece on square h1.

2.1.1.4 Examples

The initial chess position:

“rnbqkbnrpppppppp.....PPPPPPPRNBQKBNR”.

2.1.1.5 Counting the dots

To shorten the strings, subsequent dots can be replaced by a number (equal to the amount of subsequent dots).

So, an alternative for representing the initial chess position:

“rnbqkbnrpppppppp32PPPPPPPRNBQKBNR”.

An empty board: “64”.

The DLL always returns the board positions with the empty squares added together and represented by this the number.

2.2 Move representation

Moves are represented by a string describing the piece the user lifts, from what square, and the piece the user places back and on what square.

Syntax (for all moves except en passant and castling):

[PIECECODE FROM SQUARE][FROM SQUARE]
[PIECECODE TO SQUARE][TO SQUARE]

Explanation:

[PIECECODE FROM SQUARE]: a piece code of the piece on the “from square” (a character).

[FROM SQUARE]: the from square as a string: a1, b1, c1, ..., or h8.

[PIECECODE TO SQUARE]: a piece code of the piece on the “to square” (a character).

[TO SQUARE]: the to square: “a1”, “b1”, “c1”, or “h8”.

Syntax for castling moves and en passant:

[PIECECODE FROM SQUARE][FROM SQUARE]

[PIECECODE TO SQUARE][TO SQUARE]

[PIECECODE FROM SQUARE][FROM SQUARE]

[PIECECODE TO SQUARE][TO SQUARE]

Explanation:

The same as above, but with an extra **[PIECE ON FROM SQUARE][FROM SQUARE][PIECE ON TO SQUARE][TO SQUARE]** block to describe the new placement of the rook (castling) or the removal of a pawn (en passant).

2.2.1.1 Examples:

LAN notation		Driver DLL notation	
Ng1-f3	Ng8-f6	Ng1Nf3	ng8nf6
d2-d4	e7-e6	Pd2Pd4	pe7pe6
c2-c4	b7-b6	Pc2Pc4	pb7pb6
Nb1-c3	Bc8-b7	Nb1Nc3	bc8bb7
a2-a3	d7-d5	Pa2Pa3	pd7pd5
c4xd5	Nf6xd5	Pc4Pd5	nf6nd5

Special moves:

Move	Driver DLL notation
O-O white	Ke1Kg1Rh1Rf1
O-O-O black	ke8Kc8ra8rd8
promotion of a white pawn to a queen: g7xh8=Q	Pg7Qh8
a white pawn on d5 capturing a black pawn on c5 en passant: d5xc6 ep	Pd5Pc6pc5.c5 (“the white pawn on d5 becomes a white pawn on c6 and the black pawn on c5 becomes an empty square on c5)
a black pawn on e4 capturing a white pawn on f4 en passant: e4xf3 ep	pe5pf3Pf4.f4 (“the black pawn on e5 becomes a black pawn on f3 and the white pawn on f4 becomes an empty square on f4)

2.2.1.2 The move-events

The DLL triggers two move-events: WhiteMoveInput and BlackMoveInput. The moves are always semi-legal (they follow the rules of chess) but no further checking is done.

3 Events triggered by the DLL

3.1.1 Scan

When a piece is lifted or placed on the electronic board, the DLL handles this as follows:

1. The scanned position is checked for special commands, like a result input. Events are fired if such a command is detected.
2. If needed, the scanned position is rotated.
3. If the set-up dialog is visible, the new board scan is shown in it.
4. If a pointer to a custom handler function is provided, that function will be executed.

A custom scan event handling function could look like this (example in c++):

```
int __stdcall MyOnScan (char * scanned_position)
{
    //parsing the scanned board position
    int current_square = 0; //0 == square A8, 63 == square H1
    char *piece = scanned_position;
    while (*piece)
    {
        //check for empty squares...
        if (*piece >= '1' && *piece <= '6') //max. 64 empty squares
        {
            int empty_squares = *piece - '0';
            *piece++;
            if (*piece >= '0' && *piece <= '9')
            {
                empty_squares = empty_squares*10 + (*piece - '0');
                *piece++;
            }
            current_square += empty_squares;
        }
        switch (*piece)
        {
            case 'p': //a black pawn
                //your own stuff goes here
                break;
            case 'P': //a white pawn
                //your own stuff goes here
                break;
            //etc
        }
        *piece++;
        current_square++;
    }
}
```

3.1.2 Status

This event is used to trigger messages from the DLL, like version info, serial port initialisation status.

3.1.3 WClock

The WClock event fires when the white clock time on the DGT clock changes.

3.1.4 BClock

As soon as the black time on the DGT clock changes, this event is fired.

3.1.5 Result

If the user puts both kings in the centre of the board, this indicates that the game is over.

Both kings on white squares (e4, d5) means: white won.

Both kings on black squares (d4, e5) means: black won.

Any other placement: draw.

3.1.6 Newgame

Triggered as soon as all the pawns are on the initial squares and all the white pieces are on the first row, and the black ones on row number eight.

The DLL fires the newgame event with the position, to support random Fischer (chess960) set-ups and other variants.

3.1.7 WhiteMoveInput

The WhiteMoveInput event is triggered when the user lifts and places a white piece in a semi-legal way.

3.1.8 BlackMoveInput

The BlackMoveInput event is triggered when the user lifts and places a black piece in a semi-legal way.

3.1.9 WhiteMoveNow

The WhiteMoveNow event is triggered when the user lifts the white king and places it back. It is used to inform chess programs that the user wants a move NOW!

3.1.10 BlackMoveNow

The BlackMoveNow event is triggered when the user lifts the black king and places it back. It is used to inform chess programs that the user wants a move NOW!

3.1.11 StartSetup

When both kings are lifted from the board, the startsetup event is triggered.

3.1.12 StopSetupWTM

When both kings are placed back on the board, and the white one last, this event is triggered.

3.1.13 StopSetupBTM

When both kings are placed back on the board, and the black one last, this event is triggered.

4 Input to the DLL

4.1.1 WriteComPort function

This can be used to make the DLL connect to a different serial port, without using the setup dialog.

4.1.2 WritePosition function

This can be used to inform the DLL about the chess position in your program. This position is considered to be correct. If any squares are different between the position on the electronic board, and the position provided by this function, the diagram in the set-up dialog will highlight these squares. This makes it easier for the user to keep the electronic board position and the position in the chess program the same.

If a DGT XL Clock is connected, an announcement is sent to the clock. This announcement shows a maximum of two squares with the piece codes. For example, _e2 Pe4, indicating that the square e2 should be empty and a white pawn should be placed on e4.

4.1.3 WriteDebug function

In debug setting, any communication from the DLL is shown in a popup window before it is sent to your program.

4.1.4 DisplayMessage function

This function sends a message to the DGT XL clock, for some minimum amount of time.

4.1.5 EndDisplay function

This function makes the DGT XL clock go back to displaying the times after a message.

4.1.6 SetNRun function

In mode 23 of the DGT XL clock, the clock-times can be controlled with the SetNRun function.

5 Function-list

5.1.1 int __stdcall _DGTDLL_ShowDialog (int dummy);

description: shows the electronic board setup dialog

in: an integer (not used yet)

out: 1 if OK, otherwise 0.

5.1.2 int __stdcall _DGTDLL_HideDialog (int dummy);

description: hides the electronic board setup dialog

in: an integer (not used yet)

out: 1 if OK, otherwise 0.

5.1.3 int __stdcall _DGTDLL_HideDialog (int dummy);

description: hides the electronic board setup dialog, but does not unload the DLL.

in: an integer (not used yet)

out: 1 if OK, otherwise 0.

5.1.4 int __stdcall _DGTDLL_RegisterStatusFunc (int __stdcall (*statusfunc) (char *));

description: use this to let the DLL know which function to call when it fires a status event. Pass NULL (_DGTDLL_RegisterStatusFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own status function or NULL

out: 1 if OK, otherwise 0.

5.1.5 int __stdcall _DGTDLL_RegisterScanFunc (int __stdcall (*scanfunc) (char *));

description: use this to let the DLL know which function to call when it fires a scan event. Pass NULL (_DGTDLL_RegisterScanFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own scan function or NULL

out: 1 if OK, otherwise 0.

5.1.6 int __stdcall _DGTDLL_RegisterWClockFunc (int __stdcall (*wclockfunc) (char *));

description: use this to let the DLL know which function to call when it fires a wclock event. Pass NULL (_DGTDLL_RegisterWClockFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own wclock function or NULL

out: 1 if OK, otherwise 0.

5.1.7 int __stdcall _DGTDLL_RegisterBClockFunc (int __stdcall (*bclockfunc) (char *));

description: use this to let the DLL know which function to call when it fires a bclock event. Pass NULL (_DGTDLL_RegisterBClockFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own bclock function or NULL

out: 1 if OK, otherwise 0.

5.1.8 int __stdcall _DGTDLL_RegisterResultFunc (int __stdcall (*resultfunc) (char *));

description: use this to let the DLL know which function to call when it fires a result event. Pass NULL (_DGTDLL_RegisterResultFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own scan function or NULL

out: 1 if OK, otherwise 0.

5.1.9 int __stdcall _DGTDLL_RegisterNewGameFunc (int __stdcall (*newgamefunc) (char *));

description: use this to let the DLL know which function to call when it fires a newgame event. Pass NULL (_DGTDLL_RegisterNewGameFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own newgame function or NULL

out: 1 if OK, otherwise 0.

5.1.10 int __stdcall _DGTDLL_RegisterWhiteMoveInputFunc (int __stdcall (*whitemoveinputfunc) (char *));

description: use this to let the DLL know which function to call when it fires a whitemoveinput event. Pass NULL (_DGTDLL_RegisterWhiteMoveInputFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own whitemoveinput function or NULL

out: 1 if OK, otherwise 0.

5.1.11 int __stdcall _DGTDLL_RegisterBlackMoveInputFunc (int __stdcall (*blackmoveinputfunc) (char *));

description: use this to let the DLL know which function to call when it fires a blackmoveinput event. Pass NULL (_DGTDLL_RegisterBlackMoveInputFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own blackmoveinput function or NULL

out: 1 if OK, otherwise 0.

5.1.12 int __stdcall _DGTDLL_RegisterWhiteMoveNowFunc (int __stdcall (*wmn) (char *));

description: use this to let the DLL know which function to call when it fires a whitemovenow event. Pass NULL (_DGTDLL_RegisterWhiteMoveNowFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own whitemovenow function or NULL

out: 1 if OK, otherwise 0.

5.1.13 int __stdcall _DGTDLL_RegisterBlackMoveNowFunc (int __stdcall (*bmn) (char *));

description: use this to let the DLL know which function to call when it fires a blackmovenow event. Pass NULL (_DGTDLL_RegisterBlackMoveNowFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own blackmovenow function or NULL

out: 1 if OK, otherwise 0.

5.1.14 int __stdcall _DGTDLL_RegisterStartSetupFunc (int __stdcall (*startsetup) (char *));

description: use this to let the DLL know which function to call when it fires a startsetup event. Pass NULL (_DGTDLL_RegisterStartSetupFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own startsetup function or NULL

out: 1 if OK, otherwise 0.

5.1.15 int __stdcall _DGTDLL_RegisterStopSetupWTMFunc (int __stdcall (*stopsetupwtm) (char *));

description: use this to let the DLL know which function to call when it fires a stopsetupwtm event. Pass NULL (_DGTDLL_RegisterStopSetupWTMFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own scan function or NULL

out: 1 if OK, otherwise 0.

5.1.16 int __stdcall _DGTDLL_RegisterStopSetupBTMFunc (int __stdcall (*stopsetupbtm) (char *));

description: use this to let the DLL know which function to call when it fires a stopsetupbtm event. Pass NULL (_DGTDLL_RegisterStopSetupBTMFunc (NULL)) when you don't want to receive the events anymore.

in: a pointer to your own stopsetupbtm function or NULL

out: 1 if OK, otherwise 0.

5.1.17 int __stdcall _DGTDLL_WriteCOMPort (int port);

description: start the board communication on a specific serial port (1 until 8). Use this only when you don't use the dialog.

in: integer 0, 1, 2, 3, 4, 5, 6, 7 or 8

out: 1 if OK, otherwise 0.

5.1.18 int __stdcall _DGTDLL_WritePosition (char * position);

description: tells the DLL what the correct position is. For instance, when a chess program makes a move, you can call this function and send the new position to the DLL. Then, the DLL can highlight the squares that are different between the scanned position and the position in the chess program.

in: position in fen-format.

out: 1 if OK, otherwise 0.

5.1.19 int __stdcall _DGTDLL_WriteDebug (bool debug);

description: Set or unset debug mode. Debug mode pops up all communication with the DLL.

in: true or false

out: 1 if OK, otherwise 0.

5.1.20 `int __stdcall _DGTDLL_DisplayClockMessage (char * message, int time);`

description: Sends a message to display on the DGT XL Clock, for a given minimum amount of time.

in: the message and the time to display the message.

message format:

1. Character 1: **I, 1, L, |** or **space**. Any other character is handled by automatically inserting a space before it.
2. Character 2: Any character or “%+digitcode” If it is not displayable, the clock will display three lines.
3. Character 3: . (dot), , (comma), :, ;, | (for showing combining : and .) or space. Any other character is handled by automatically inserting a space before it.
4. Character 4: Any character or “%+digitcode” If it is not displayable, the clock will display three lines.
5. Character 5: Any character or “%+digitcode” If it is not displayable, the clock will display three lines.
6. Character 6: A space. Any other character is handled by automatically inserting a space before it.
7. Character 7: Like character 1.
8. etc.

With the option to insert %+digitcode e.g. (%1, %64 ...) in the message, to make special characters, you can build up the digits yourself. The code is a sum of the segments of the display (see diagram 2). For example, to make a small ‘u’, you need to add up 16+8+4 and you can add this in the message as “%28”. Of course, it’s easier to insert a small ‘u’ directly in the string: message “1u-” is the same as message “%6%28%64”.

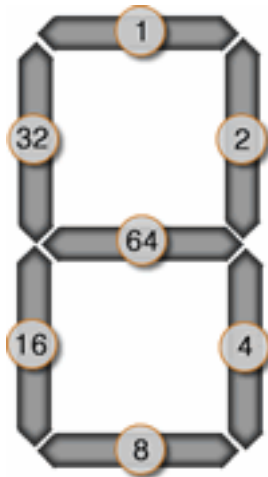


Diagram 2

out: 0.

5.1.21 `int __stdcall _DGTDLL_EndDisplay (int c);`

description: ends to display a message on the DGT XL clock.

in: a dummy variable c. It can be any value.

out: 1.

5.1.22 `int __stdcall _DGTDLL_RegisterStableBoardFunc (int __stdcall (*func) (char *));`

description: use this to let the DLL know which function to call when it fires a stable board event. Pass NULL (`_DGTDLL_RegisterStableBoardFunc (NULL)`) when you don't want to receive the events anymore. The event is fired when the board position did not change for a certain time-period (the stabletime).

in: a pointer to your own stableboard function or NULL

out: 1 if OK, otherwise 0.

5.1.23 `int __stdcall _DGTDLL_SetNRun (char * wclock, char * bclock, int runwho);`

description: Controls the clock-times of a DGT-XL clock, in mode 23.

in: wclock and bclock, strings of 7 characters long: h:mm:ss.

Runwho: 0: clock set the times and pause.

1: white clock will countdown. Black clock will pause

2: black clock will countdown. White clock will pause

3: both the white and black clock will countdown.

out: 1 if OK, otherwise 0.

5.1.24 `int __stdcall _DGTDLL_ClockMode (int nothing)`

description: Checks if the clock might is in mode 23.

in: a dummy variable

out: 23 if the clock is in mode 23, otherwise 0.

5.1.25 `int __stdcall _DGTDLL_SetAutoRotation (bool autorotate);`

description: Enable or disable auto rotation.

in: True: autorotation on. False: autorotation off

out: 1 if ok, otherwise 0.