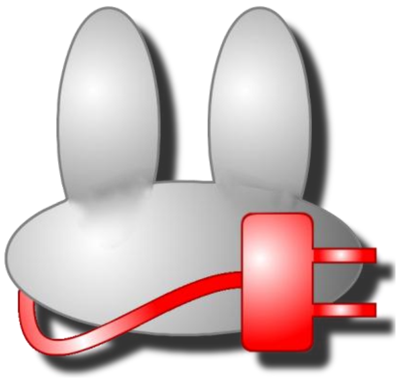
DGT Electronic Board DLL API

Programmer's manual



1. Table of contents

2. Version history

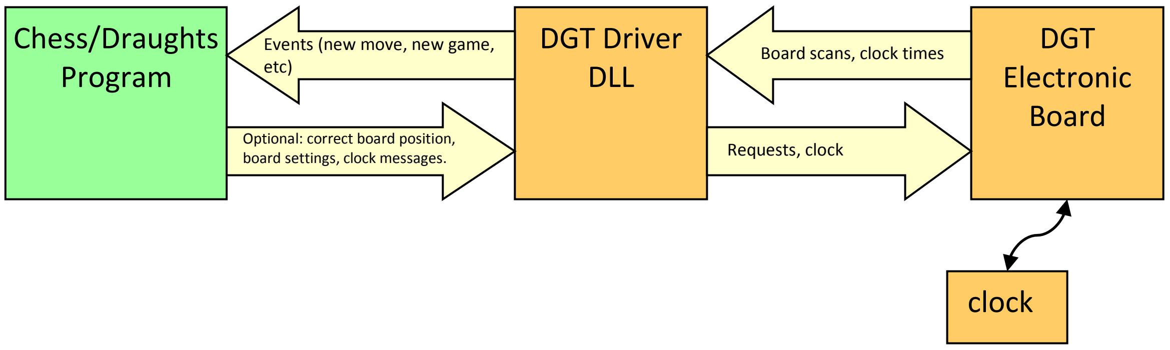
2.01 (Jan 18, 2011)
Some minor changes. PlayWhiteMove/PlayBlackMove will only support SAN format.

2.0 Draft (Jan 4, 2011)
RabbitPlugin description added. Added piece codes for magic pieces. Choice between PluginFen and FEN. Code example rewritten.

1.20 (Nov 20, 2003)
Last document version describing the ‘old’ (1.x) driver

3. Overview of DGT Electronic Board DLL API

The DGT Electronic Board DLL acts as a layer between the DGT electronic board and the (chess or draughts) program. It reads the positions and clock times from the board and converts them to events.
For every event, the chess program can (optionally) add a pointer from the DLL to a function in the program that handles the event. All events generated by the DLL are described in chapter 6.
The addition of these pointers to custom event-handling functions is done by "register" functions. For example, the DLL function "_DGTDLL_RegisterScanFunc". Chapter 8 contains a list of all the functions in the DLL, which are described in more detail in chapter 10.
Besides these register functions, the DLL has a built-in dialog, where the user can choose the serial port and see the current board position.



a. RabbitPlugin

The 1.x driver was just called "DGT Electronic Board DLL. From the end of 2011, this is (gradually) replaced by a new DLL, called RabbitPlugin, which carries version 2.x. RabbitPlugin is a complete rewritten version of the DLL, based on the Rabbit library, and so it supports the same board games (currently two chess and five draughts variants) and uses the same reconstructor as RabbitQueen and other Rabbit applications.

The interface of the new DLL has been extended with a number of functions, but it is backwards compatible with the 1.x driver.

b. wxWidgets

RabbitPlugin makes use of wxWidgets for the dialog and event processing structure. An important consequence of this is that when the master application also uses wxWidgets, both the application and RabbitPlugin must make use of the same shared wxWidgets libraries. Currently, wxWidgets 2.9.3 is being used; contact us if your application also uses wxWidgets to keep an eye on compatibility. The wxWidgets shared library files (wxbase293u_vc_custom.dll and the others) are installed in the System directory. You can retrieve the wxWidgets version that the DLL uses with the function _DGTDLL_GetWxWidgetsVersion().

c. Event loop

The board driver DLL handles window events, which is driven by an event loop that the application must provide. A consequence is that the driver doesn't work directly in a console application; you either have to use a GUI, or manually add an event loop that processes window events.

d. Location, 64 bit, platforms

On Windows, filename is **dgtebdll.dll**, it is 32 bit and installed in the System32 directory. The other directories it is installed in, is only for backwards compatibility with Chessbase and should not be used anymore in future programs. Just look for "dgtebdll.dll" in your application without any path prefix, because the System directories are already in the PATH variable.

A 64 bit version is planned to become available in 2012. The name will be **dgtebdll64.dll** .

Linux/Mac support is also planned in 2012. The Linux name will probably be **dgtebdll.so** (and **dgtebdll64.so** for 64bit). The Mac OSX library name will probably be **libdgtebdll.dylib**.

In all variants, the function names are the same and always start with _DGTDLL_ , regardless of the platform.

e. Debugging

For application developers, a debug version of RabbitPlugin is available. Its installer gives you two extra choices that are useful in debugging:

- Debug configuration. If checked, then the debug configuration of dgtebdll.dll is installed instead of the release version. If you run the application in debug configuration from a suitable debugging environment, all runtime errors in the DLL or in the application are caught and passed to the debugger with detailed information.
- EmptyDLL. If checked, then a stripped version of dgtebdll.dll is installed instead of the normal one. Its dialog is a blank dialog and it contains no communication or reconstruction logic. The underlying structure is the same (i.e. it uses wxWidgets in the same way as the normal DLL) and it does some logging. Logging can be set in the normal DLL, then install the EmptyDLL over it. This is useful to isolate problems in loading/initializing the DLL.

This debug version is not directly available on the website, but you can contact us for a download link.

4. Board and move representation

a. Board representation: PluginFen and FEN

The application has the choice to use either PluginFen or FEN for communicating the board from/to the DLL. Default is PluginFen (which the 1.x driver uses); to switch to FEN you can call the function _DGTDLL_UseFEN(true). When playing draughts, it is recommended to use FEN because this is more standard.

b. PluginFen description

General description of PluginFen

A board is described by a single string, similar to (but different from) the the Forsyth Edwards Notation (FEN). Every square is represented by a single character in the string. If the board game is Chess, 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.

Piece codes

Chess:

- **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
- **X** for magic piece indicating draw result (PluginFen only, not in FEN)
- **Y** for magic piece indicating white win (PluginFen only, not in FEN)
- **Z** for magic piece indicating black win (PluginFen only, not in FEN)

Draughts:

- **M** for a white man, **m** for a black man
- **K** for a white king, **k** for a black king
- **X**, **Y**, **Z** for the magic pieces as above (PluginFen only, not in FEN)

Empty Square:

An empty square is represented by a dot (".") or the digit 1 (meaning: one empty square). Multiple consecutive empty squares can be collected together and be represented by a single number.

Examples

The initial chess position, in PluginFen:
"rnbqkbnrpppppppp................................PPPPPPPPRNBQKBNR" (which is equivalent to "rnbqkbnrpppppppp32PPPPPPPPRNBQKBNR")

An empty board: "64".

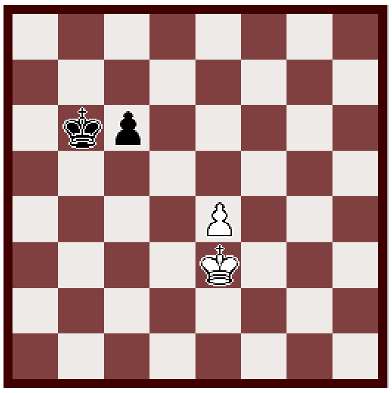


Diagram 1: 17kp17P7K

The diagram above: “**17kp17P7K19**”. (Meaning: 17 empty squares, a black king, a black pawn, 17 empty squares, a white pawn, 7 empty squares, a white king, 19 empty squares).

When communicating using PluginFen, The DLL may return the board positions with the empty squares added together and represented by this the number.

c. Move representation: PluginMove and SAN

There are two formats for the moves: PluginMove and SAN. They are used for both incoming and outgoing move communications.
The default format is PluginMove; you can switch to SAN by calling the function _DGTDLL_UseSAN(true).

PluginMove

PluginMove 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).

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 capturing a rook: 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	pe4pf3Pf4.f4 (“the black pawn on e4 becomes a black pawn on f3 and the white pawn on f4 becomes an **empty square** on f4)
Chess960 castling; originally king at f1, rook at g1	Kf1Kg1Rg1Rf1
Takback of g7xh8=Q (which captured a rook)	Qh8Pg7.h8rh8

SAN

SAN is the format that is also used in PGN/PDN files.
Examples for International Chess: e4, Nf3, Rac1, N3e4, Qb1d4, O-O, hxg8=Q+, no extra annotations allowed.

If a takeback is recorded, then a separate callback function is called.

International Draughts uses numeric notation: 11-15, 23-18, 10x19. English Checkers use numbers 1-32; Brazilian/Russian draughts and Turkish checkers use algebraic notation (a1-h8). See the most applicable PGN/PDN/SAN specification for the details. Please contact us when you plan to implement draughts communications with this DLL so that we can check if all conforms to the latest specifications and decide on open issues (if any).

The move-events

The DLL triggers two move-events: WhiteMoveInput and BlackMoveInput. The moves are always semi-legal (they follow the rules of the currently active board game). In driver 1.x, no further checking is done. In driver 2.x, more legality checks are performed, so it is important that the right board game is chosen!

Default format is PluginMove; if the function _DGTDLL_UseSAN(true) was called then the format switches to SAN.

If the user is allowed to takeback moves, then these may be triggered as well. If white plays Ng1-f3, it triggers WhiteMoveInput(PluginMove: "Ng1Nf3"; SAN: "Nf3") and it is Black's turn. If White now takebacks the last move, then WhiteMoveInput is triggered again: WhiteMoveInput(PluginMove: "Nf3Ng1"; SAN: "Takeback")

5. Displaying text on the clock

If the clock is in mode 23, the times and countdown mode can be set in the SetNRun function. The clock automatically counts down the running clocks and sends the updated clock times to the application.

When a move or position is sent to the DLL to tell the computer's move to the user, the DLL computes the difference between the last scan position and the 'goal' position, and shows it automatically on the clock's display.

Besides these functions, it is possible to set a custom text on the clock's display. The text is passed as a string and the format is as follows:

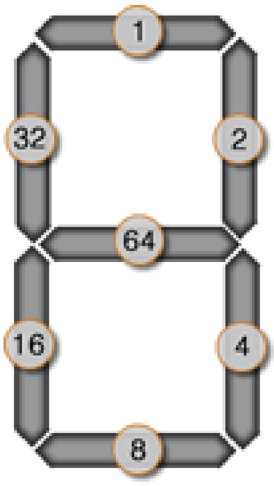
```
1A:BC 1D:EF
```

Both sides (left&right) have five available character positions. Three of them have a 7-segment display capable of showing most alphanumeric characters; the other two positions have only 1 or 2 segments. The 6th position is the "space" between the clock sides, so that the string has 11 positions in total. Every position is either a single character, or a %+digitcode. Sometimes part of the string is shifted to the right, if one is not displayable on a certain position.

1. Position 1: **I**, **1**, **L**, **|** or **space.** Any other character is handled by automatically inserting a space before it.
2. Position 2: Any character or "%+digitcode" If it is not displayable, the clock will display three lines.
3. Position 3: **.** (dot), **,** (comma), **:**, **;**, **|** (for showing combining **:** and **.**) or space. Any other character is handled by automatically inserting a space before it.

4. Position 4: Any character or "%+digitcode" If it is not displayable, the clock will display three lines.
5. Position 5: Any character or "%+digitcode" If it is not displayable, the clock will display three lines.
6. Position 6: A space. Any other character is handled by automatically inserting a space before it.
7. Position 7: Like character 1.
8. Like position 2.
9. Like position 3.
10. Like position 4.
11. Like position 5.

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".



Change since 2.0: At most three digits after the "%" are parsed, so if you use all three digits (pad with zeroes if necessary), then you can use normal digits for the next positions.

6. Events triggered by the DLL

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 be defined like this (example in c++):

```
int __stdcall MyScanCallback (char* scanned_position) { }
```

See the Partial C++ example chapter for a more details.

2. Status

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

3. WClock

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

4. BClock

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

5. Result

If board detects a result, the result is sent to the application. See “Rabbit Commons User Manual” about how the result is detected (kings, magic pieces).

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.

7. WhiteMoveInput

The WhiteMoveInput event is triggered when the user lifts and places a white piece in a semi-legal way. If the format is PlayMove, then takebacks (if allowed) are sent by this callback function as well. If format is SAN, then a separate callback function is called.

8. BlackMoveInput

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

9. WhiteTakeback

Is triggered when the move format is SAN, takebacks are allowed, and white takes back the last move.

10. BlackTakeback

Is triggered when the move format is SAN, takebacks are allowed, and black takes back the last move.

11. 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!

12. 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!

13. StartSetup

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

14. StopSetupWTM

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

15. StopSetupBTM

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

16. GameTypeChanged

Is triggered when the board game changes.

17. AllowTakebacksChanged

Is triggered when the allow-takebacks setting changes.

18. MagicPiece

Is triggered when one of the three magic pieces is placed somewhere on the board. See ‘Rabbit Commons User Manual’ about which electronic boards recognize magic pieces.

7. Input to the DLL

Return value: 0 means success; nonzero means error. This is consistent with the behaviour of the 1.x driver, but the 1.x version of this document stated something different. This has now been corrected.

Note that usually only limited checking is done, because most processing is done in different threads. So a function might return 0 (success), but still cause an error

1. WriteComPort function

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

2. WritePosition function

This can be used to inform the DLL about the chess position in your program. Both PluginFen and FEN are supported, but please use FEN only after calling _DGTDLL_UseFEN(1).

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, Pe4 _e2 , indicating that a white pawn should be placed on e4 and that the square e2 should be empty.

3. PlayWhiteMove / PlayBlackMove

This is the other way to inform the DLL about the move made by the application. The format is PluginMove, unless _DGTDLL_UseSAN(true) was called earlier, which sets the format to SAN.

If PlayWhiteMove is called while it is White's turn in the DLL, and the move is playable, then the user is instructed to execute that move (by highlighting different squares, by a spoken announcement and/or by setting the clock display).

If PlayWhiteMove is called while it is NOT White's turn in the DLL, then the last white move is compared with the PlayWhiteMove argument. If it is equal, then this is considered to be already performed and thus ignored. If it is different, then the user is instructed to takeback the last move and play the correct white move instead.

4. WriteDebug function

In debug setting, any communication from the DLL is shown in a popup window before it is send to your program. (This is now obsolete; the log gives much more information)

5. DisplayMessage function

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

6. EndDisplay function

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

7. SetNRun function

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

8. Function overview

Here is a list with the functions in C++ notation (C users can omit the `const` keyword).

```
typedef int __stdcall FC(const char*);
typedef int __stdcall FI(int);
typedef int __stdcall FB(bool);
typedef int __stdcall F();
typedef int __stdcall FIIC(int, int, const char*);

extern "C" {
	int __stdcall _DGTDLL_GetVersion();
	int __stdcall _DGTDLL_GetWxWidgetsVersion();
	int __stdcall _DGTDLL_Init();
	int __stdcall _DGTDLL_Exit();
	int __stdcall _DGTDLL_ShowDialog(int);
	int __stdcall _DGTDLL_HideDialog(int);
	int __stdcall _DGTDLL_WriteCOMPort(int);
	int __stdcall _DGTDLL_WriteCOMPortString(const char*);
	int __stdcall _DGTDLL_WritePosition(const char*);
	int __stdcall _DGTDLL_PlayWhiteMove(const char*);
	int __stdcall _DGTDLL_PlayBlackMove(const char*);
	int __stdcall _DGTDLL_WriteDebug(bool);
	int __stdcall _DGTDLL_DisplayClockMessage(const char*,int);
	int __stdcall _DGTDLL_EndDisplay(int);
	int __stdcall _DGTDLL_SetNRun(const char*,const char*,int);
	int __stdcall _DGTDLL_ClockMode(int);
	int __stdcall _DGTDLL_SetAutoRotation(bool);
	int __stdcall _DGTDLL_UseFEN(bool);
	int __stdcall _DGTDLL_UseSAN(bool);
	int __stdcall _DGTDLL_SetGameType(int);
	int __stdcall _DGTDLL_AllowTakebacks(bool);

	int __stdcall _DGTDLL_RegisterStatusFunc(FC*);
	int __stdcall _DGTDLL_RegisterScanFunc(FC*);
	int __stdcall _DGTDLL_RegisterStableBoardFunc(FC*);
	int __stdcall _DGTDLL_RegisterWClockFunc(FC*);
	int __stdcall _DGTDLL_RegisterBClockFunc(FC*);
	int __stdcall _DGTDLL_RegisterResultFunc(FC*);
	int __stdcall _DGTDLL_RegisterNewGameFunc(FC*);
	int __stdcall _DGTDLL_RegisterWhiteMoveInputFunc(FC*);
	int __stdcall _DGTDLL_RegisterBlackMoveInputFunc(FC*);
	int __stdcall _DGTDLL_RegisterWhiteTakebackFunc(F*);
	int __stdcall _DGTDLL_RegisterBlackTakebackFunc(F*);
	int __stdcall _DGTDLL_RegisterWhiteMoveNowFunc(FC*);
	int __stdcall _DGTDLL_RegisterBlackMoveNowFunc(FC*);
	int __stdcall _DGTDLL_RegisterStartSetupFunc(FC*);
	int __stdcall _DGTDLL_RegisterStopSetupWTMFunc(FC*);
	int __stdcall _DGTDLL_RegisterStopSetupBTMFunc(FC*);
	int __stdcall _DGTDLL_RegisterGameTypeChangedFunc(FI*);
	int __stdcall _DGTDLL_RegisterAllowTakebacksChangedFunc(FB*);
	int __stdcall _DGTDLL_RegisterMagicPieceFunc(FIIC*);
}
```

9. Partial C++ example

The listing below shows how you can initialize the DLL and set a callback function. It uses the standard windows mechanism for loading a DLL at runtime, but it does NOT include a suitable event loop and so the driver dialog won't actually show up yet.

```
#include <Windows.h>

typedef int __stdcall F();
typedef int __stdcall FI(int);
typedef int __stdcall FB(bool);
typedef int __stdcall FC(const char*);
typedef int __stdcall FCC(const char*, const char*);
typedef int __stdcall FFC(FC*);

int __stdcall myScanCallback(const char* scan) {
  // Scan received
  return 0;
}

void example() {
  HINSTANCE hDll = LoadLibrary("dgtebdll.dll");
  if (hDll==nullptr) {
    // dgtebdll.dll not found
  } else {
    F* getVersionFunc = reinterpret_cast<F*>(GetProcAddress(hDll, "_DGTDLL_GetVersion"));
    F* getWxWidgetsVersionFunc = reinterpret_cast<F*>(GetProcAddress(hDll,
"_DGTDLL_GetWxWidgetsVersion"));
    F* initFunc = reinterpret_cast<F*>(GetProcAddress(hDll, "_DGTDLL_Init"));
    F* exitFunc = reinterpret_cast<F*>(GetProcAddress(hDll, "_DGTDLL_Exit"));
    FFC* registerScanFunc = reinterpret_cast<FFC*>(GetProcAddress(hDll, "_DGTDLL_RegisterScanFUnc"));
    FI* showDialogFunc = reinterpret_cast<FI*>(GetProcAddress(hDll, "_DGTDLL_ShowDialog"));

    if (getVersionFunc!=nullptr) {
      int version = getVersionFunc();
    } else {
      // version 1.x
    }
    // if you use wxWidgets, you should compare getWxWidgetsVersionFunc() with wxVERSION_NUMBER
    if (initFunc!=nullptr) {
      int value = initFunc();
      if (value!=0) {
        // 1 = already initialized; 2 = error
      }
    }
    // now the DLL is properly loaded and initialized
    if (registerScanFunc!=nullptr) {
      registerScanFunc(&myScanCallback);
      // From now on, myScanCallback is called when new scans come in
    }
    //////////// register other callback functions

    //////////// your stuff here, keep in mind that you still need an event loop

    // exiting
    if (exitFunc!=nullptr) {
      int value = exitFunc();
      if (value!=0) {
        // non-clean exit
      }
    }
    int success = FreeLibrary(hDll);
    if (!success) {
      // get error code with GetLastError()
    }
    // now the DLL is properly de-initialized and unloaded
  }
}
```

10. Function-list

int __stdcall _DGTDLL_GetVersion();

Description: Returns the version of the DLL.
Returns: A 6 digit number in the form of *xxyyzz*. (If fewer digits are returned, pad it with zeroes.) *xx* is the major version number, *yy* is the minor number, *zz* is the release number. 2.0.x are debug releases. Example: return value of 20006 = 020006 = 2.0.6 (= 2.0 Beta 6). The EmptyDLL returns 0. Calling this function does not auto-initialize the library.
Since: 2.0.7 . If the function is not present, the version is 1.x

int __stdcall _DGTDLL_GetWxWidgetsVersion();

Description: Returns the version of wxWidgets as used in the DLL; returns the macro wxVERSION_NUMBER.
Returns: A 4 digit number in the form of *xyzz*, (see also wx/version.h). Example: 2903 = wxWidgets version 2.9.3. The EmptyDLL also returns the correct wxWidgets version. Calling this function does not auto-initialize the library, so if your application uses wxWidgets, you can use this function to check if the versions are the same. If not, then you probably have to inform the user to upgrade the application or RabbitPlugin to match versions.
Since: 2.0.7 . If the function is not present, then the driver doesn't use wxWidgets at all.

int __stdcall _DGTDLL_Init();

Description: This initializes the DLL explicitly. Calling this function is voluntary. If not called, then the DLL is automatically initialized when necessary (i.e. in the first call of most functions).
Returns: 0 if successful, 1 if the DLL was already initialized (which can be ignored), 2 if something went seriously wrong and DLL cannot be used.
Since: 2.0.7 . If function is not present, it is initialized automatically.

int __stdcall _DGTDLL_Exit();

Description: This de-initializes the DLL explicitly. Calling this function is recommended, as this makes it easier to terminate threads cleanly, but it is not necessary. If not called, then the DLL is automatically de-initialized when the DLL is unloaded.
Returns: 0 if successful, 1 if something went wrong (which can be ignored).
Since: 2.0.7 . If function is not present, then the DLL is de-initialized automatically.

int __stdcall _DGTDLL_ShowDialog(int dummy);

Description: Shows the electronic board setup dialog.
Parameter: irrelevant
Return: 0 if OK, otherwise 1.
Since: 1.0

int __stdcall _DGTDLL_HideDialog(int dummy);

Description: Hides the electronic board setup dialog, but does not unload the DLL.
Parameter: irrelevant
Returns: 0 if OK, otherwise 1.
Since: 1.0

int __stdcall _DGTDLL_WriteCOMPort(int port);

Description: connect to a specific serial port. Use this only if you have a serial port selection dialog from which the user chose one, because the RabbitPlugin driver can already automatically re-connect to the last used serial port.
Parameter: any positive integer. The DLL forms a port descriptor by appending the value to the "COM" word. E.g. a value of 12 becomes COM12.
Returns: 0 if OK, otherwise 1. Actually, it is not checked whether the port really exists.
Since: 1.0

int __stdcall _DGTDLL_WriteCOMPortString(char* port);

Description: connect to a specific serial port; this version is not restricted to COM* ports only.
Parameter: a port descriptor, e.g. "COM12" or "/dev/ttyS1"
Returns: 0 if OK, otherwise 1. Actually, it is not checked whether the port descriptor is a valid string and the port really exists.
Since: 2.0.6

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.
Parameter: Position in FEN or PluginFen format. Default format is PluginFen; you can switch to normal FEN by calling _DGTDLL_UseFEN(1). If the wrong format is used, then a warning appears in the logfile and processed according to the normal format. When using FEN, in case of Chess/Chess960, the parameter must consist of either one field (the board), or six fields seperated by spaces.
Returns: 0 if OK, otherwise 1.
Since: 1.0

int __stdcall _DGTDLL_PlayWhiteMove(char* move);

Description: Tells the DLL the next white move. Format must be SAN format and UseSAN(true) must be active.
Parameter: White move in SAN format.
Returns: 0 if OK, 1 if useSAN(true) was not yet called, 2 on another error.
Since: 2.0.9?

int __stdcall _DGTDLL_PlayBlackMove(char* move);

Description: Tells the DLL the next black move. Format is PluginMove, unless _DGTDLL_UseSAN(true) was called.
Parameter: Black move in PluginMove or SAN format.
Returns: 0 if OK, 1 if useSAN(true) was not yet called, 2 on another error.
Since: 2.0.9?

int __stdcall _DGTDLL_WriteDebug(bool debug);

Description: Set or unset debug mode. Debug mode pops up all communication with the DLL. It is probably easier to use the log instead.
Parameter: 1 to enable pop-ups, 0 to disable.
Returns: 0 if OK, otherwise 1.
Since: 1.0

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.
Parameter message: The message; see “Displaying text on the clock”
Parameter time: The time in milliseconds. After the time has passed, the display is returned to showing times.
Returns: 0 if OK, otherwise 1.
Since: 1.0

int __stdcall _DGTDLL_EndDisplay(int dummy);

Description: Clears the message (if any), setting the display back to showing clock times.
Parameter: irrelevant
Returns: 0 if OK, otherwise 1.
Since: 1.0

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

Description: Controls the clock-times of a DGT-XL clock, in mode 23.
Parameter: 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.
Returns: 0 if OK, otherwise 1.
Since: 1.0

int __stdcall _DGTDLL_ClockMode(int dummy);

Description: Intends to check if the clock is in mode 23, but is not implemented in the board anyway.

Parameter: irrelevant
Returns: 23 if the clock is in mode 23, otherwise 0; But don't rely on this result.
Since: 1.0

int __stdcall _DGTDLL_SetAutoRotation(bool autorotate);

Description: Enable or disable auto rotation.
Parameter: true: autoration on. False: autoration off.
Returns: 0 if ok, otherwise 1.
Since: 1.0

int __stdcall _DGTDLL_UseFEN(bool value);

Description: Sets position format.
Parameter: true: use FEN. False: use PluginFen. Initial value is PluginFen.
Returns: 0 if ok, otherwise 1.
Since: 2.0.7

int __stdcall _DGTDLL_UseSAN(bool value);

Description: Sets movetext format.
Parameter: true: use SAN. False: use Pluginmove. Initial value is PluginMove.
Returns: 0 if ok, otherwise 1.
Since: 2.0.7

int __stdcall _DGTDLL_SetGameType(int gameType);

Description: Sets game type.
Parameter: See Wiki/PDN for the latest code specification. Current: 0=Chess, 10=Chess960, 20=International Draughts 20x20, 21=English Draughts, 25=Russian Draughts, 26=Brazilian Draughts, 29=Turkish Checkers
Returns: 0 if ok, otherwise 1.
Since: 2.0.10?

int __stdcall _DGTDLL_AllowTakebacks(bool allowed);

Description: Sets if takebacks are allowed.
Parameter: true if allowed, false if not allowed.
Returns: 0 if ok, otherwise 1.
Since: 2.0.10?

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own status function or NULL
Returns: 0 if OK, otherwise 1.

Since: 1.0

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 when you don’t want to receive the events anymore.
Parameter: a pointer to your own scan function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0

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 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 stable time), which can be changed by the user in the dialog.
Parameter: a pointer to your own stable board function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0

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 when you don’t want to receive the events anymore.
Parameter: a pointer to your own wclock function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0

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 when you don’t want to receive the events anymore.
Parameter: a pointer to your own bclock function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0

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 when you don’t want to receive the events anymore.
Parameter: a pointer to your own scan function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0, 2.0.8

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

Description: use this to let the DLL know which function to call when it fires a new game event. Pass NULL when you don't want to receive the events anymore.
Parameter: a pointer to your own new game function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own whitemoveinput function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own blackmoveinput function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.0

int __stdcall _DGTDLL_RegisterWhiteTakebackFunc (int _stdcall (*whiteTakebackFunc) ());

Description: use this to let the DLL know which function to call when it fires a WhiteTakeback event. Pass NULL when you don't want to receive the events anymore. The event is only triggered when the move format is SAN and takebacks are allowed.
Parameter: a pointer to your own WhiteTakeback function or NULL
Returns: 0 if OK, otherwise 1.
Since: 2.0.9?

int __stdcall _DGTDLL_RegisterBlackTakebackFunc (int _stdcall (*whiteTakebackFunc) ());

Description: use this to let the DLL know which function to call when it fires a BlackTakeback event. Pass NULL when you don't want to receive the events anymore. The event is only triggered when the move format is SAN and takebacks are allowed.
Parameter: a pointer to your own BlackTakeback function or NULL
Returns: 0 if OK, otherwise 1.
Since: 2.0.9?

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own whitemovenow function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.x, 2.0.8

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own blackmovenow function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.x, 2.0.8

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own startsetup function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.x, 2.0.8

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own scan function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.x, 2.0.8

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 when you don't want to receive the events anymore.
Parameter: a pointer to your own stopsetupbtm function or NULL
Returns: 0 if OK, otherwise 1.
Since: 1.x, 2.0.8

int __stdcall _DGTDLL_RegisterGameTypeChangedFunc (int _stdcall (*myGameTypeChangedFunc) (int));

Description: Sets a callback function which is called when the game type has changed, by the user or programmatically. If your application doesn't support it, you can overrule the change by a call to _DGTDLL_SetGameType(int), but not from within the callback function.
Parameter: a pointer to your own GameTypeChanged function or NULL. The int parameter of the callback function gives the GameType.
Returns: 0 if OK, otherwise 1.
Since: 2.0.10?

int __stdcall _DGTDLL_RegisterAllowTakebacksChangedFunc (int _stdcall (*myAllowTakebacksChangedFunc) (bool));

Description: Registers a callback function which is called when the "allow takebacks" setting has changed, by the user or programmatically. If your application doesn't allow it, you can overrule it by a call to _DGTDLL_AllowTakebacks(bool) , but not from within the callback function.
Parameter: a pointer to your own AllowTakebacksCahanged function or NULL. The bool parameter of the callback function gives the current takebacks setting.
Returns: 0 if OK, otherwise 1.
Since: 2.0.10?

int __stdcall _DGTDLL_RegisterMagicPieceFuncFunc (int _stdcall (*myMagicPieceFunc) (int x, int y, char* piece));

Description: Registers a callback function which is called when a magic piece is placed on the board. The callback function must have three parameters: *x*, *y*, *piece*. (*x*,*y*) define the coordinate on the board of the magic piece, (0,0) is the bottom-left corner of the board regardless of the board game. So (4,3) defines field e4 on a 8x8 board. The *piece* parameter indicates the magic piece type: "X"=half-white/half-black, "Y"= white, "Z"=black.
Parameter: a pointer to your own magic piece function or NULL.
Returns: 0 if OK, otherwise 1.
Since: 2.0.9?

11. Contact

Support: support@dgtprojects.com
Information: info@dgtprojects.com
Website: www.dgtprojects.com