Jörg Kuthe

qtConsole

Instructions for the Use of the qtConsole Software

Revision date: 25th of February 2011

QT software GmbH
Konstanzer Str. 10
D-10707 Berlin
Germany

Telefon 030/9290087-0
Telefax 030/9290087-2

http://www.qtsoftware.de
eMail info@qtsoftware.de

Contents

1. Introduction

Many Fortran programmers still use the console (under Windows this is also called the DOS command window, DOS box, or DOS window) as their primary output device, because it is simple to use and the Fortran language is designed to use it for output to screen (via WRITE(*,...), PRINT) and input from the keyboard (via READ(*,...)). In fact, this is not only true for Fortran, but also for many other standardized programming languages. However, most other programming language implementations under Windows offer the programmer a lot of extensions to be able to create graphical user interfaces (GUIs), but these are not standardized and thus mostly not portable to other operating systems. To create GUI based programs is possible in Fortran too, but not well supported, and it requires some good knowledge about the Windows Application Interface (WinAPI) and much more programming efforts than programming the output to the console. Thus, for Fortran programmers the preferred "device" for output is the console. However the simplicity of using Fortran's console output concept implies several limitiations. For example:

- output using a single color only (usually white text on black background)
- output is written consecutively (free text positioning is often difficult)
- no control on the cursor's position, size and visibility
- no control on the position and size of the console window relative to the desktop screen (in case of GUI based operating systems)
- no control on the number of lines being buffered in the console buffer

There are certainly other limitations (to name a few: font size is fixed, no graphics, no mouse control) but the ones listed above are those the qtConsole library addresses. By supplying a few functions which are very easy to use, the Fortran programmer is enabled to use for his console output some coloring and positioning facilities, for example to highlight important text or to structure the output to console.

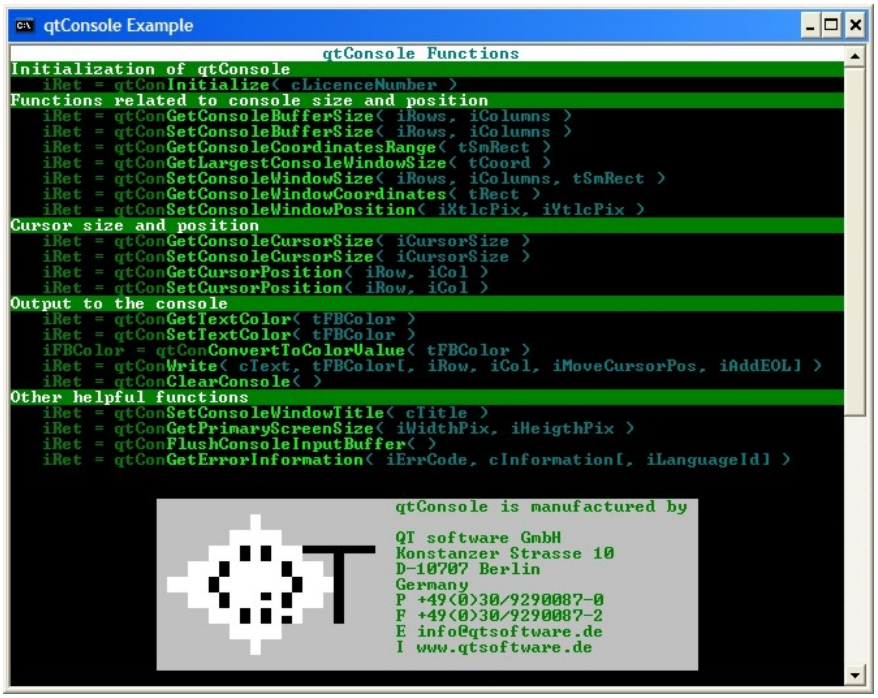

Fig. 1: Screen shot showing the list of qtConsole's functions (see sample Ex01.exe)

2. List of Functions

The following gives you an overview of the functions in qtConsole:

Initialization

```
iRet = qtConInitialize( cLicenceNumber )
! to be called one-time and before any other call to
  a qtConsole function
```

Functions related to console size and position

```
iRet = qtConGetConsoleBufferSize( iRows, iColumns )
! returns the size of the internal console buffer
  (in rows, columns).

iRet = qtConSetConsoleBufferSize( iRows, iColumns )
! set the size (rows, columns) of the internal
  console buffer.

iRet = qtConGetConsoleCoordinatesRange( tRect )
! returns the coordinates of the upper-left and
  lower-right corners of the console window

iRet = qtConGetLargestConsoleWindowSize(
            tLargestConsoleWindowSize )
! returns the size of the largest possible console
  window based on the current font and the size of
  the display

iRet = qtSetChangeConsoleWindowSize( iRows,
            iColumns, tRect )
! changes the size of the console window

iRet = qtConGetConsoleWindowCoordinates( tFrame )
! returns size and position of console window (in
  pixels relative to desktop coordinate system)

iRet = qtConSetConsoleWindowPosition( iXtlcPix,
            iYtlcPix )
! positions the console window by specification of
  its top left corner (tlc) in pixels

iRet = qtConGetPrimaryScreenSize( iWidthPix,
            iHeigthPix )
! returns the width and height of the client area
  for a full-screen window on the primary display
  monitor
```

Cursor

A console uses a cursor (something like a blinking underscore) which is not identical to the one you usually see on the Windows screen (the "arrow" you position by moving the mouse).

```
iRet = qtConGetCursorPosition( iRow, iCol )
! returns the current position of the cursor

iRet = qtConSetCursorPosition( iRow, iCol )
! positions the cursor

iRet = qtConGetConsoleCursorSize( iCursorSize )
! returns the current size of the cursor

iRet = qtConSetConsoleCursorSize( iCursorSize )
! changes the size of the cursor (may also affect
  its visibility)
```

Output to the console

qtConsole allows to control the fore- and background colors being used for writing. The functions may be called in combination with FORTRAN's PRINT and WRITE(*,...) commands.

```
iRet = qtConGetTextColor( tFBColor )
! gets the current color being used for following
  write operations

iRet = qtConSetTextColor( tFBColor )
! sets the color for following write operations

iRet = qtConWrite( cText, tFBColor[, iRow, iCol,
            iMoveCursorPosition, iAddEOL] )
! writes text to the console using the specified
  color and optionally positions the cursor before
  writing and optionally adding linefeeds

iFBColor = qtConConvertToColorValue( tFBColor )
! converts a color specification to an INTEGER value

iRet = qtConClearConsole( )
! clears the console window contents
```

Other helpful functions

```
iRet = qtConSetConsoleWindowTitle( cTitle )
! sets the title (caption) of the console window

iRet = qtConFlushConsoleInputBuffer( )
! flushes the console input buffer

iRet = qtConGetErrorInformation( iErrCode,
            cInformation[, iLanguageId] )
! returns error information for an error code being
  specified (optionally a language Id can be given
  too)
```

3. General Guidelines

Naming

Looking at the list of function (above) you will notice that the names of the functions and their arguments conform to several rules which make usage much easier. These are:

- All qtConsole functions are `FUNCTIONs` returning an integer value (`iRet`). If its value is 0, than the function performed successful. Otherwise an error occurred and an error code is returned as the function's value (in `iRet`).
- Their names begin with the prefix `qtCon`.
- Dummy argument names begin with a prefix indicating their type.
  `i`: `INTEGER` type
  `c`: `CHARACTER` type
  `t`: derived `TYPE`

Coordinates System and Units

Console's coordinates are rows and columns. The row and column count start at value 0. The origin (row = 0, column = 0) of the console coordinates system is the top left corner (tlc) of the console window. Rows are counted vertically positive from top to the bottom of the console

window (default range is usually: 0 to 23). Columns are counted horizontally positive from left to right (default range is usually: 0 to 79). So, the default coordinates of the bottom right corner (brc) are usually (23,79).
Window's coordinates are usually measured in pixels. The window coordinates system has the same orientation as the console coordinates system: tlc is (0,0), brc is (screen width-1, screen height-1).
Unless specified otherwise, the console coordinates system is being referred to.

Programming

The qtConsole functions and several Fortran `KINDs`, `TYPEs` and `PARAMETERs` are made available via a Fortran MODULE named qtConsole. Thus, the first line in your Fortran source code should be:

```
USE qtConsole
```

Then, qtConsole has to be initialized using the licence number you have received. For example:

```
iRet = qtConInitialize('L2487-123456')
```

In case a qtConsole function failed (`iRet /= 0`), you may call `qtConGetErrorInformation` to obtain the meaning of the error code.

```
iRet = qtConInitialize('L1234-567890')
IF ( iRet /= 0 ) THEN
  ! an error occurred
  iRet = qtConGetErrorInformation( iRet,  &
              cInformation )
  ! ...
END IF
```

Licence File (qtSetLicence_QTCONSOLE, in qtSetLicence_L2847_######.f90)

Internally, `qtConInitialize` calls a routine named `qtSetLicence_QTCONSOLE`. qtConsole initially provides a file named `qtSetLicence_QTCONSOLE.f90` which supplies a dummy routine for evaluation. When you receive your licence, you also receive a file whose name conforms to `qtSetLicence_L2847_######.f90` (with `#` being a digit, and `L2847_######` being your qtConsole licence number). This file contains a

```
SUBROUTINE qtSetLicence_QTCONSOLE( iError )
```

which replaces the dummy you may have used before. Add `qtSetLicence_L2847_######.f90` to your project, such that it is compiled an linked.

Compiling & Linking

Because the qtConsole `MODULE` is being used, your compiler requires the specification of the `MODULE` path (the directory where the `qt*.mod` files for your compiler is located).
When linking, the linker is searching for the qtConsole functions that are used in your program. It will find them in the qtConsole library (`qtConsole*.lib` file) suitable for your compiler & linker system. You may have to give its full pathname to the linker.

Royalties - Distribution of your Program

For the distribution of your program (.exe) that uses qtConsole there are no royalties to be paid for the usage of qtConsole when you have purchased a licence of qtConsole.

4. Reference

4.1 KINDs, TYPEs and PARAMETERs

The Fortran MODULE qtConsole provides KINDs, TYPE definitions and PARAMETERs that are used by qtConsole functions. Internally they are defined in a MODULE named qtConsoleKindsTypes whose contents you see below. Since qtConsole uses qtConsoleKindsTypes these definitions are accessible (PUBLIC).

```
MODULE qtConsoleKindsTypes
   USE qtConsoleKindsCompiler ! compiler specific KINDs
   IMPLICIT NONE

   ! Kinds & Types
   INTEGER, PARAMETER :: qt_K_BYTE = Selected_INT_Kind(2)    ! INTEGER*1
   INTEGER, PARAMETER :: qt_K_SHORT = Selected_INT_Kind(4)   ! INTEGER*2
   INTEGER, PARAMETER :: qt_K_LONG  = Selected_INT_Kind(8)   ! INTEGER*4
   !
   INTEGER, PARAMETER :: qt_K_INT = qt_K_LONG
   INTEGER, PARAMETER :: qt_K_SINT = qt_K_LONG
   INTEGER, PARAMETER :: qt_K_UINT = qt_K_LONG  ! not quite correct, but
                                 ! there is no unsigned INTEGER in Fortran
   INTEGER, PARAMETER :: qt_K_WORD  = qt_K_SHORT
   INTEGER, PARAMETER :: qt_K_DWORD = qt_K_LONG
   INTEGER, PARAMETER :: qt_K_BOOL  = qt_K_LONG

   INTEGER, PARAMETER :: qt_K_SCHAR = qt_K_BYTE   ! = 1

   INTEGER, PARAMETER :: qt_K_HANDLE = qtConsoleCompiler_PointerLength
   ! from qtConsoleCompiler (32-Bit: INTEGER*4, 64-Bit: INTEGER*8)
   INTEGER, PARAMETER :: qt_K_LPVOID = qtConsoleCompiler_PointerLength

   INTEGER, PARAMETER :: qt_I_Black       =  0,  &
                         qt_I_DarkBlue    =  1,  &
                         qt_I_Green       =  2,  &
                         qt_I_Turquoise   =  3,  &
                         qt_I_Red         =  4,  &
                         qt_I_Violet      =  5,  &
                         qt_I_Olive       =  6,  &
                         qt_I_LightGrey   =  7,  &
                         qt_I_Grey        =  8,  &
                         qt_I_Blue        =  9,  &
                         qt_I_LightGreen  = 10,  &
                         qt_I_LightBlue   = 11,  &
                         qt_I_LightRed    = 12,  &
                         qt_I_LightViolet = 13,  &
                         qt_I_Yellow      = 14,  &
                         qt_I_White       = 15

   INTEGER, PARAMETER :: qt_I_Error_Offset = 106000
   INTEGER, PARAMETER :: qt_I_Error_INVALID_COLOR_VALUE =  &
                                          qt_I_Error_Offset + 7
   INTEGER, PARAMETER :: qt_I_Error_WRITE_FAULT = qt_I_Error_Offset + 29

   TYPE qt_T_COORD
      SEQUENCE
      INTEGER(qt_K_SHORT) :: x
      INTEGER(qt_K_SHORT) :: y
   END TYPE

   TYPE qt_T_SMALL_RECT
      SEQUENCE
      INTEGER(qt_K_SHORT) :: Left
      INTEGER(qt_K_SHORT) :: Top
      INTEGER(qt_K_SHORT) :: Right
      INTEGER(qt_K_SHORT) :: Bottom
   END TYPE

   TYPE qt_T_RECT
      SEQUENCE
      INTEGER(qt_K_LONG) :: left
      INTEGER(qt_K_LONG) :: top
      INTEGER(qt_K_LONG) :: right
      INTEGER(qt_K_LONG) :: bottom
   END TYPE

   TYPE qt_T_CONSOLE_SCREEN_BUFFER_INFO
      SEQUENCE
      TYPE (qt_T_COORD)      :: dwSize
      TYPE (qt_T_COORD)      :: dwCursorPosition
      INTEGER(qt_K_WORD)     :: wAttributes
      TYPE (qt_T_SMALL_RECT) :: srWindow
      TYPE (qt_T_COORD)      :: dwMaximumWindowSize
   END TYPE
```

The MODULE qtConsoleKindsTypes - continued

```
   TYPE qt_T_CONSOLE_CURSOR_INFO
      SEQUENCE
      INTEGER(qt_K_DWORD) :: dwSize
      INTEGER(qt_K_BOOL)  :: bVisible
   END TYPE

   TYPE qt_T_FBColor
      SEQUENCE
      INTEGER(qt_K_BYTE) :: iForegroundColor
      INTEGER(qt_K_BYTE) :: iBackgroundColor
   END TYPE

END MODULE qtConsoleKindsTypes
! (C) Copyright by Jörg Kuthe, QT software GmbH, Germany.
!     All rights reserved. 2011
! -------------------------------------------------------
! http:\\www.qtsoftware.de        email: qt@qtsoftware.de
! =======================================================
```

The MODULE qtConsoleKindsTypes (part of MODULE qtConsole)

4.2 qtConsole Functions

qtConClearConsole

FUNCTION qtConClearConsole( )

```
INTEGER (qt_K_WORD) :: qtConClearConsole
```

Clears the contents of the console and positions the cursor at the top left corner (0,0).

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet

! clear the console screen
iRet = qtConClearConsole( )
```

qtConConvertToColorValue

FUNCTION qtConConvertToColorValue(tFBColor)

```
INTEGER (qt_K_WORD) :: qtConConvertToColorValue
TYPE (qt_T_FBColor), INTENT(IN) :: tFBColor
```

The foreground and background colors are usually specified by variables of `TYPE (qt_T_FBColor)`. To convert them into an `INTEGER` value, the function `qtConConvertToColorValue` is supplied. The conversion is carried out according to

```
f = tFBColor % iForegroundColor  ! values 0 - 15
b = tFBColor % iBackgroundColor  ! values 0 - 15
iFBColor = b * 16 + f
```

If an invalid color is specified in the argument `tFBColor`, `qtConConvertToColorValue` return the value -1.

Example

```
USE qtConsole
TYPE (qt_T_FBColor) :: tFBColor
INTEGER (qt_K_WORD) :: iFBColor

tFBColor % iForegroundColor = qt_I_Red   ! = 4
tFBColor % iBackgroundColor = qt_I_White ! = 15
```

```
iFBColor = qtConConvertToColorValue( tFBColor )
! iFBColor = 15*4 + 4 = 64
```

qtConFlushConsoleInputBuffer

FUNCTION qtConFlushConsoleInputBuffer( )

```
INTEGER (qt_K_DWORD) :: qtConFlushConsoleInputBuffer
```

Calling `qtConFlushConsoleInputBuffer` causes all input records currently in the input buffer to be discarded. This effectively means that a user is not able to "type ahead". If you want to make sure that a user is typing in at the time a READ statement is executed, call `qtConFlushConsoleInputBuffer`.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
CHARACTER(1) cYN
! make sure that the user sees the prompt and any
! ahead typing is ignored
iRet = qtConFlushConsoleInputBuffer( )
PRINT*, 'Enter y (for yes) or n (for no):'
READ(*, '(A)') cYN
```

qtConGetConsoleBufferSize

FUNCTION qtConGetConsoleBufferSize(iRows, iColumns)

```
INTEGER(qt_K_DWORD) :: qtConGetConsoleBufferSize
INTEGER(qt_K_SHORT), INTENT(OUT) :: iRows
INTEGER(qt_K_SHORT), INTENT(OUT) :: iColumns
```

As you probably have noticed, a console is usually a window that is scrollable. That implies, there are rows that are buffered. In fact, a console internally uses a character buffer, which may contain up to 300 rows with each having 80 columns, for example. To determine the number of rows and columns the console buffer can contain, call qtConGetConsoleBufferSize.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
INTEGER (qt_K_SHORT) :: iRows, iColumns
! determine the number of rows and columns of the
! console's internal buffer
iRet = qtConGetConsoleBufferSize( iRows, iColumns )
```

qtConGetConsoleCoordinatesRange

FUNCTION qtConGetConsoleCoordinatesRange(tRect)

```
INTEGER(qt_K_DWORD) :: qtConGetConsoleCoordinatesRange
TYPE (qt_T_SMALL_RECT), INTENT(OUT) :: tRect
```

The console's row and column coordinates of the upper-left and lower-right corners are returned by `qtConGetConsoleCoordinatesRange`.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
TYPE (qt_T_SMALL_RECT) :: tRect
INTEGER (qt_K_SHORT) :: iRowMax, iColMax
! determine the maximum values for row and column
! coordinates (minimum values are usually (0,0))
iRet = qtConGetConsoleCoordinatesRange( tRect )
iRowMax = tRect % Bottom
iColMax = tRect % Right
```

qtConGetConsoleCursorSize

FUNCTION qtConGetConsoleCursorSize(iCursorSize)

```
INTEGER(qt_K_DWORD) :: qtConGetConsoleCursorSize
INTEGER (qt_K_BYTE), INTENT(OUT) :: iCursorSize
```

The size of the console's cursor (the "blinking underscore") can be determined by calling `qtConGetConsoleCursorSize`. The size can vary from 0 and 100. A value of 0 means that the cursor is not visible.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
INTEGER (qt_K_BYTE) :: iCursorSize

iRet = qtConGetConsoleCursorSize( iCursorSize )
IF ( iCursorSize == 0 ) THEN
  ! cursor invisible
  ! ...
END IF
```

qtConGetConsoleWindowCoordinates

FUNCTION qtConGetConsoleWindowCoordinates(tFrame)

```
INTEGER(qt_K_DWORD) ::
                    qtConGetConsoleWindowCoordinates
TYPE (qt_T_RECT), INTENT(OUT) :: tFrame
```

The size and position of console window in pixels relative to desktop coordinates system is returned by `qtConGetConsoleWindowCoordinates`.
The top left corner's (tlc) coordinates are
(`tFrame % top`, `tFrame % left`),
and the bottom right corner's (brc) coordinates are
(`tFrame % bottom`, `tFrame % right`). So, the width and height of the console window can be calculated (see the example). Note that the origin of the desktop coordinates system is (0,0).

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
TYPE (qt_T_RECT) :: tFrame
INTEGER(qt_K_LONG) :: iWidthPix, iHeightPix

iRet = qtConGetConsoleWindowCoordinates( tFrame )
```

```
iWidthPix = tFrame % right - tFrame % left
iHeightPix = tFrame % bottom - tFrame % top
```

qtConGetCursorPosition

FUNCTION qtConGetCursorPosition(iRow, iCol)

```
INTEGER(qt_K_DWORD) :: iRet
INTEGER (qt_K_SHORT), INTENT(OUT) :: iRow, iCol
```

The position of the console's cursor (the "blinking underscore") can be determined by calling `qtConGetCursorPosition`.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
INTEGER (qt_K_SHORT) :: iRow, iCol
INTEGER iErr, iNum
!...

WRITE(*, 6000, ADVANCE='NO')
6000 FORMAT('Enter a valid number: ')
! get the input position
iRet = qtConGetCursorPosition( iRow, iCol )
DO
  READ (*, 5000, IOSTAT=iErr) iNum
  IF ( IsValid( iNum ) ) EXIT
  ! input is invalid, position the cursor at input pos.
  iRet = qtConSetCursorPosition( iRow, iCol )
END DO
```

qtConGetErrorInformation

FUNCTION qtConGetErrorInformation(iErrCode, cInformation[, iLanguageId])

```
INTEGER(qt_K_DWORD) :: qtConGetErrorInformation
INTEGER(qt_K_DWORD), INTENT(IN) :: iErrCode
CHARACTER(*), INTENT(OUT) :: cInformation
INTEGER(qt_K_DWORD), INTENT(IN),  &
                     OPTIONAL :: iLanguageId
```

If a qtConsole function fails, its return value is an error codes whose meaning can be determined by calling `qtConGetErrorInformation`. The meaning is usually returned in the operating system's language unless the argument `iLanguageID` (optional) is specified. For available language identifiers see the WinAPI function `FormatMessage`.
The meaning being returned in argument `cInformation` may contain formatting characters (e.g. linefeeds).
The length of the character argument (`cInformation`) necessary for retrieving the error code meaning is unknown but less than 64K bytes. Usually a few hundred bytes is sufficient.

Example

```
SUBROUTINE ErrorHandling( iErr, cErrPos )
   USE qtConsole
   IMPLICIT NONE
   INTEGER(qt_K_DWORD), INTENT(IN) :: iErr
   CHARACTER(*), INTENT(IN) :: cErrPos
   !
```

```
   CHARACTER(200) :: cInformation
   INTEGER iRet

   IF ( iErr == 0 ) RETURN

   iRet = qtConGetErrorInformation( iErr,   &
                                    cInformation )
   IF ( iRet == 0 ) THEN
      WRITE(*,6000) iErr, TRIM(cInformation),   &
                    TRIM(cErrPos)
      6000 FORMAT('Error ', I0, ': ', A / &
                  'Occurred at ', A)
   ELSE
      WRITE(*,6090) iErr, TRIM(cErrPos)
      6090 FORMAT('Error ', I0, ', no further', &
           ' information available', &
           (qtConGetErrorInformation failed).' / &
           'Occurred at ', A)
   END IF
END SUBROUTINE ErrorHandling
```

qtConGetLargestConsoleWindowSize

FUNCTION qtConGetLargestConsoleWindowSize(tLargestConsoleWindowSize)

```
INTEGER(qt_K_DWORD) ::
                    qtConGetLargestConsoleWindowSize

TYPE (qt_T_COORD), INTENT(OUT) ::
                              tLargestConsoleWindowSize
```

This function returns the size of the largest possible console window, based on the current font and the size of the display (in rows, columns).
The maximum values of the coordinates for the lower-right corner of the console window are the returned values in tLargestConsoleWindowSize less one:

```
   tRect % Bottom = tLargestConsoleWindowSize % Y - 1
   tRect % Right = tLargestConsoleWindowSize % X - 1
```

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
TYPE (qt_T_COORD) :: tLCWS
INTEGER (qt_K_SHORT) :: iRowMax, iColMax
! determine the maximum possible values for row and
! column coordinates
iRet = qtConGetLargestConsoleWindowSize( tLCWS )
iRowMax = tLCWS % Y - 1
iColMax = tLCWS % X - 1
```

qtConGetPrimaryScreenSize

FUNCTION qtConGetPrimaryScreenSize(iWidthPix, iHeightPix)

```
INTEGER(qt_K_DWORD) :: qtConGetPrimaryScreenSize

INTEGER(qt_K_INT), INTENT(OUT) :: iWidthPix

INTEGER(qt_K_INT), INTENT(OUT) :: iHeigthPix
```

The width and height (in pixels) of the client area for a full-screen window on the primary display monitor is returned by `qtConGetPrimaryScreenSize`. The client area is usually a window area less its frame and caption. But since a desktop or full screen

respectively does not have a frame or caption, `qtConGetPrimaryScreenSize` should return the screen's resolution.

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet
INTEGER(qt_K_INT) :: iWidthPix, iHeigthPix

iRet = qtConGetPrimaryScreenSize( iWidthPix,   &
                                  iHeigthPix )
```

qtConGetTextColor

FUNCTION qtConGetTextColor(tFBColor)

```
INTEGER(qt_K_DWORD) :: qtConGetTextColor
TYPE (qt_T_FBColor), INTENT(OUT) :: tFBColor
```

Internally, the console uses a default color for any following write operation. To obtain this color call `qtConGetTextColor`.
The foreground color is to be found in the argument's member `tFBColor % iForegroundColor` and the background color in `tFBColor % iBackgroundColor`.

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet
TYPE (qt_T_FBColor) :: tFBColor

iRet = qtConGetTextColor( tFBColor )
IF ( tFBColor % iForegroundColor    &
      == tFBColor % iBackgroundColor ) THEN
  ! the output is invisible
  ! ...
END IF
```

qtConInitialize

FUNCTION qtConInitialize(cLicenceNumber)

```
INTEGER(qt_K_DWORD) :: qtConInitialize
CHARACTER (*), INTENT(IN) :: cLicenceNumber
```

Before any other usage of a qtConsole function, call `qtConInitialize`. It requires a single argument, the access code which can either be `'evaluation'` or a licence number (the one you have received) which refers to the licence file `qtSetLicence_L2847_######.f90` (with # being a digit, and `L2847_######` being your licence number) that has to be linked in.
Note: If your compiler runtime system doesn't open a console window on startup of your program, this function might fail. If this happens, insert a `WRITE(*,...)` statement before the call of `qtConInitialize`.

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet

iRet = qtConInitialize('evaluation')
! allows qtConsole to run in evaluation mode
```

```
! The access code 'evaluation' is to be replaced by
! the licence number you receive when you acquire
! qtConsole, for example:
! iRet = qtConInitialize('L2847-790412')
```

qtConSetConsoleBufferSize

FUNCTION qtConSetConsoleBufferSize(iRows, iColumns)

```
INTEGER(qt_K_DWORD) :: qtConSetConsoleBufferSize
INTEGER(qt_K_SHORT), INTENT(IN) :: iRows
INTEGER(qt_K_SHORT), INTENT(IN) :: iColumns
```

A console window manages its scrollable contents using an internal character buffer. Its size can be changed using qtConSetConsoleBufferSize.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
INTEGER (qt_K_SHORT) :: iRows, iColumns
! get console buffer size and double the number of
! rows being buffered
iRet = qtConGetConsoleBufferSize( iRows, iColumns )
iRows = iRows * 2
iRet = qtConSetConsoleBufferSize( iRows, iColumns
```

qtConSetConsoleCursorSize

FUNCTION qtConSetConsoleCursorSize(iCursorSize)

```
INTEGER(qt_K_DWORD) :: qtConSetConsoleCursorSize
INTEGER (qt_K_BYTE), INTENT(IN) :: iCursorSize
```

The size of the console's cursor (the "blinking underscore") can be changed using qtConSetConsoleCursorSize. The cursor size can vary from 0 to 100. If the cursor's size is 0, the cursor is invisible.

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet
INTEGER (qt_K_BYTE) :: iDefCurSize

! get the default cursor size
iRet = qtConGetConsoleCursorSize( iDefCurSize)
! ...
! make the cursor invisible
iRet = qtConSetConsoleCursorSize( 0_qt_K_BYTE )
! ...
! restore the default cursor size
iRet = qtConSetConsoleCursorSize( iDefCurSize )
```

qtConSetConsoleWindowPosition

FUNCTION qtConSetConsoleWindowPosition(iXtlcPix, iYtlcPix)

```
INTEGER(qt_K_DWORD) :: qtConSetConsoleWindowPosition
INTEGER(qt_K_INT), INTENT(IN) :: iXtlcPix,
```

```
INTEGER(qt_K_INT), INTENT(IN) :: iYtlcPix
```

To position the console window relative to the desktop coordinates system call `qtConSetConsoleWindowPosition` and specify the console window's coordinates of the top left corner (tlc) in pixel coordinates.

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet
TYPE (qt_T_RECT) :: tWndRect
INTEGER (qt_K_INT) :: iXPix, iYPix, iWidthPix,
iHeightPix

! center the console window on the screen
iRet = qtConGetConsoleWindowCoordinates( tWndRect )
iRet = qtConGetPrimaryScreenSize( iWidthPix,   &
                                  iHeightPix )
iXPix = ( iWidthPix -
          (tWndRect % right - tWndRect % left) ) / 2
iYPix = ( iHeightPix -
          (tWndRect % bottom - tWndRect % top) ) / 2
iRet = qtConSetConsoleWindowPosition( iXPix, iYPix )
```

qtConSetConsoleWindowSize

`FUNCTION qtConSetConsoleWindowSize( iRows, iColumns, tRect )`

```
INTEGER (qt_K_DWORD) :: qtConSetConsoleWindowSize
INTEGER(qt_K_SHORT), INTENT(IN) :: iRows
INTEGER(qt_K_SHORT), INTENT(IN) :: iColumns
TYPE (qt_T_SMALL_RECT), INTENT(OUT) :: tRect
```

The function changes the size of the console window. The requested size is specified by the arguments `iRows` and `iColumns`. This requested size may not be possible due to the limitations imposed by the size of the desktop screen and the current console font. In this case the possible maximum value for either rows or columns or both is used.
The new console's window coordinates range is returned by argument `tRect` whose members `tRect % Bottom` and `tRect % Right` contain the maximum value of the row and column coordinate, respectively. Note that the row and column count start at value 0.

Note

The change of the window size does not take place immediately. Windows handle this asynchonously. This means, the function may return before the change of size has already occurred. Thus, calling any function that relies on the new size of the window, has to check before whether the new size of the window has already been achieved.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
TYPE (qt_T_SMALL_RECT) :: tRect

! Try to change console's window size to 70 rows,
! 120 columns.
iRet = qtConSetConsoleWindowSize( 70_qt_K_SHORT,  &
                                  120_qt_K_SHORT, tRect )
! Because the function requires the first to arguments
```

```
! to be of INTEGER(qt_K_SHORT) type and KIND, the
! values are extended by the appropriate KIND
! specification.
! The actual size is dependent on the minimum size
! allowed by the system. The console's new coordinates
! range is returned in tRect.
```

qtConSetConsoleWindowTitle

FUNCTION qtConSetConsoleWindowTitle(cTitle)

```
INTEGER(qt_K_DWORD) :: qtConSetConsoleWindowTitle
CHARACTER(*), INTENT(IN) :: cTitle
```

When a console application is started, a console window pops up and its title (caption) is usually the name of the executable. This function allows to change title of the console window. Simply specify a title text in argument `cTitle`.

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet

iRet = qtConSetConsoleWindowTitle( 'qtConsole Example'
)
```

qtConSetCursorPosition

FUNCTION qtConSetCursorPosition(iRow, iCol)

```
INTEGER(qt_K_DWORD) :: iRet
INTEGER (qt_K_SHORT), INTENT(IN) :: iRow, iCol
```

The console's cursor (the "blinking underscore") can be positioned by calling `qtConSetCursorPosition`. The position is specified by the arguments `iRow`, `iCol` in the console coordinates system with its origin (0, 0) being at the top left corner.

Example

```
USE qtConsole
INTEGER (qt_K_DWORD) :: iRet
! position the cursor at the beginning of row 10
iRet = qtConSetCursorPosition( 10_qt_K_SHORT,   &
                               0_qt_K_SHORT )
```

qtConSetTextColor

FUNCTION qtConSetTextColor(tFBColor)

```
INTEGER(qt_K_DWORD) :: qtConSetTextColor
TYPE (qt_T_FBColor), INTENT(IN) :: tFBColor
```

To change the console's default color being used for any following write operations, call qtConSetTextColor.
The foreground color is to be specified in the argument's member `tFBColor % iForegroundColor` and the background color in `tFBColor % iBackgroundColor`.

Note: When setting the color for following `WRITE(*,...)` or `PRINT` to console operations, consider that your compiler runtime system may use an

internal buffer for subsequent output operations. Usually, this buffer is output to the console only when a linefeed occurs (advancing output) or a `READ(*,...)` operation follows. Therefore the following example will probably not work, when non-advancing output is buffered.

```
6000 FORMAT(A)
WRITE (*, 6000, ADVANCE='NO') 'For example, '   &
                //' you might want to highlight some '
   ! non-advancing WRITE being buffered
iRet = qtConSetTextColor( tColorHighlight )
WRITE (*, 6000, ADVANCE='NO') 'important text'
   ! non-advancing WRITE being buffered
iRet = qtConSetTextColor( tColorDefault )
WRITE (*, 6000) '.' ! advancing WRITE (causes output to
   ! console), but the change of the color is ignored
```

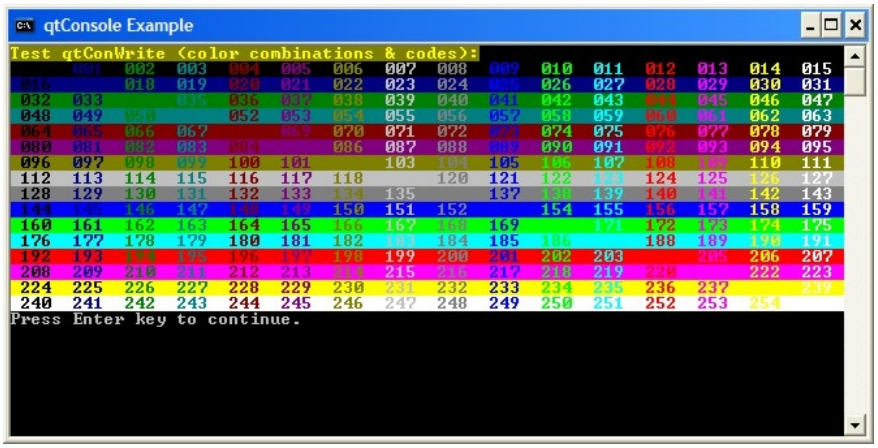

Fig. 2: Screen shot showing color codes (see sample Ex01.exe)

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet
TYPE (qt_T_FBColor) :: tFBColor

tFBColor % iForegroundColor = qt_I_Blue
tFBColor % iBackgroundColor = qt_I_LightGrey
iRet = qtConSetTextColor( tFBColor )
```

qtConWrite

FUNCTION qtConWrite(cText, tFBColor[,
iRow, iCol,
iMoveCursorPosition,
iAddEOL])

```
INTEGER(qt_K_DWORD) :: qtConWrite
CHARACTER (*), INTENT(IN) :: cText
TYPE (qt_T_FBColor), INTENT(IN) :: tFBColor
INTEGER (qt_K_SHORT), INTENT(INOUT), OPTIONAL :: iRow
INTEGER (qt_K_SHORT), INTENT(INOUT), OPTIONAL :: iCol
INTEGER (qt_K_SHORT), INTENT(IN),
                      OPTIONAL :: iMoveCursorPosition
INTEGER (qt_K_SHORT), INTENT(IN), OPTIONAL :: iAddEOL
```

qtConsole provides two ways to write a colored text to the console. One way is to set a text color (calling `qtConSetTextColor`) and then using Fortran WRITE(*,...) or PRINT statements to write to the console. The other possibility is offered by this function which has the advantage against Fortran WRITE and PRINT that no linefeed is necessary to be

output. Specify the text to be written in the argument `cText` and the fore- and background colors in the argument `tFBColor`. Then, `qtConWrite` outputs the text to the console and the cursor is moved to the character position after the text being written unless any of the optional arguments specifies something else.
If the optional argument `iMoveCursorPosition` is `0`, the text is written starting either at the current cursor position or at the position specified by the optional arguments `iRow` and `iCol`. Then the cursor is re-positioned at the first character of the text being output.
If optional argument `iMoveCursorPosition` is not equal to `0`, the cursor is moved as the text is written (default behaviour).
If optional arguments `iRow` and `iCol` are specified, the text is written starting at the cursor position given and the cursor's position after output is returned. This may be the same as specified on input, when the optional argument `iMoveCursorPosition` specifies a value of `0`.
If `iMoveCursorPosition` is not specified or if and its value is `/= 0`, the optional argument `iAddEOL` can be used to add a number of line feeds after writing the text. If `iAddEOL=0` no linefeed is output (default behaviour).

Example

```
USE qtConsole
INTEGER(qt_K_DWORD) :: iRet
TYPE (qt_T_FBColor) :: tFBColor
INTEGER (qt_K_BYTE) :: iFGColor, iBGColor
CHARACTER (5) ::  cAtt
! output all color combinations to the console
DO iBGColor = 0, 15
  tFBColor % iBackgroundColor = iBGColor
  DO iFGColor = 0, 15
    tFBColor % iForegroundColor = iFGColor
    iFBColor = qtConConvertToColorValue( tFBColor )
    WRITE (cAtt, 10000) iFBColor
    10000 FORMAT(1X, I3.3, 1X)
    iRet = qtConWrite( cAtt, tFBColor )
  END DO
END DO
```

5. Compile & Link

5.1 General Notes

To generate programs (.exe) that use qtConsole generally the following requirements are needed.

Compiler Requirements

The `MODULE` path (the directory where the `qt*.mod` files for your compiler is located) needs to be found by your compiler.

Either your licence file `qtSetLicence_L2847_######.f90` (with # being a digit, and `L2847_######` being your qtConsole licence number) or the dummy licence file `qtSetLicence_QTCONSOLE.f90` (provided for evaluation purposes) have to be added to your project and have to be compiled.

Linker Requirements

The qtConsole library (`qtConsole*.lib` file) suitable for your compiler & linker system needs to be linked in. You may have to give its full pathname to the linker.
The operating systems libraries `kernel32.lib`, `user32.lib` and `advapi32.lib` need to be linked too. With some compilers this happens automatically, others require them to be specified explicitely and then, they are usually supplied with your compiler system.

Sample Projects

It is tried to provide you with ready-to-use projects that you can easily load into your IDE (integrated development environment). With some IDEs this may not be possible, because these do not allow to save relative file references (but absolute file paths). If this is the case there will be a note, that you have to change some project settings to make them work properly.

5.2 With Compaq Visual Fortran (CVF)

The following files are supplied for usage with Compaq Visual Fortran (abbreviated CVF):

```
qtconsole.mod
qtconsolecompiler.mod
qtconsolekindscompiler.mod
qtconsolekindstypes.mod
qtconsolevfinterfaces.mod
qtdatetime.mod
qtdatetimekindstypes.mod
qtConsoleCVF.lib
```

The last one is the static library, the other ones are compiled `MODULE`s. They all are found in a subdirectory named `CVF`.
There is also a sample project file (`Ex01.dsp`) and its workspace file (`Ex01.dsw`) in the subdirectory `CVF\Ex01`. You may start CVF's Developer Studio by loading the workspace file. The project file contains all

necessary settings to generate an executable from the example Ex01.f90. Just start the "Build" process.

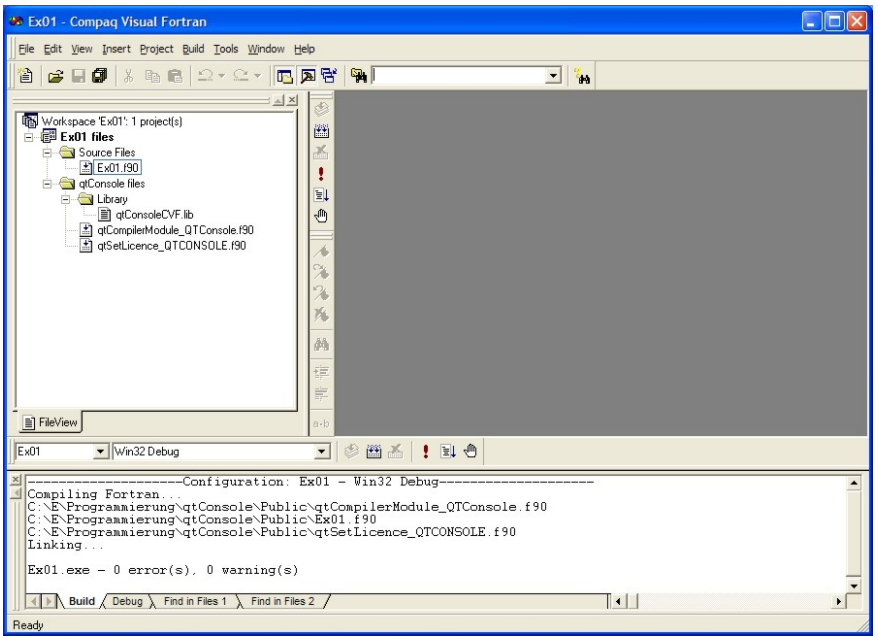

Fig. 3: The Ex01 workspace and project in CVF's Developer Studio.

The screen shot above shows three qtConsole files

```
qtConsoleCVF.lib
qtCompilerModule_QTCONSOLE.f90
qtSetLicence_QTCONSOLE.f90
```

being part of the project. The latter one is the dummy licence file that you have to replace by your licence file `qtSetLicence_L2847_######.f90` (with `######` to be replaced by your qtConsole licence number). These three files have to be added to any of your projects that use qtConsole functions.

Project Settings

- Fortran preprocessor:
  - If the `qt*.mod` files are not in the project's directory, specify a `USE` path for qtConsole `MODULE`s (files `qt*.mod`). See the screen shot for an example.
- Link input:
  - Specify the Windows system library modules
    `kernel32.lib`
    `user32.lib`
    `advapi32.lib`.

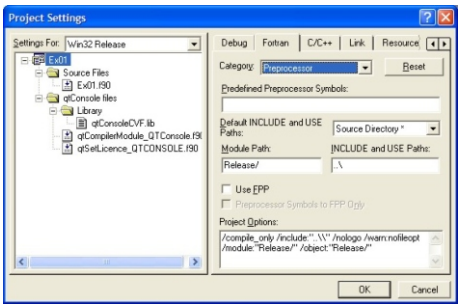

Fig. 4: Setting the USE path such that CVF finds the qt.mod files.*

- To get rid of the warning `LINK : warning LNK4098: defaultlib "libc.lib" conflicts with use of other libs; ...` which can occur in Debug mode, you may specify to ignore `libc.lib`.

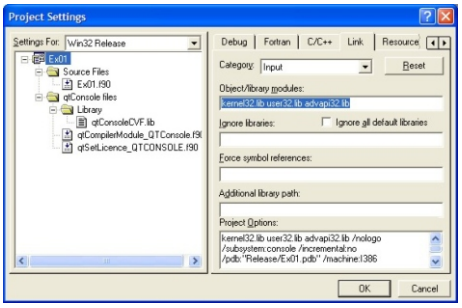

Fig. 6: Setting the link input such that the required Windows system libraries are linked.

5.3 With Intel Visual Fortran (IVF)

The following files are supplied for usage with Intel Visual Fortran (abbreviated IVF) for creation of 32-bit executables:

```
qtconsole.mod
qtconsolecompiler.mod
qtconsolekindscompiler.mod
qtconsolekindstypes.mod
qtconsolevfinterfaces.mod
qtdatetime.mod
qtdatetimekindstypes.mod
qtConsoleIVF.lib
```

The last one is the static library, the other ones are compiled `MODULE`s. They all are found in a subdirectory named `IVF`.
There is also a sample project file (`Ex01.vfproj`) and its solution file (`Ex01.sln`) in the subdirectory `IVF\Ex01`. You may start IVF's Visual Studio by loading the solution file. The project file contains all necessary settings to generate an executable from the example Ex01.f90. Just start the "Build" process.

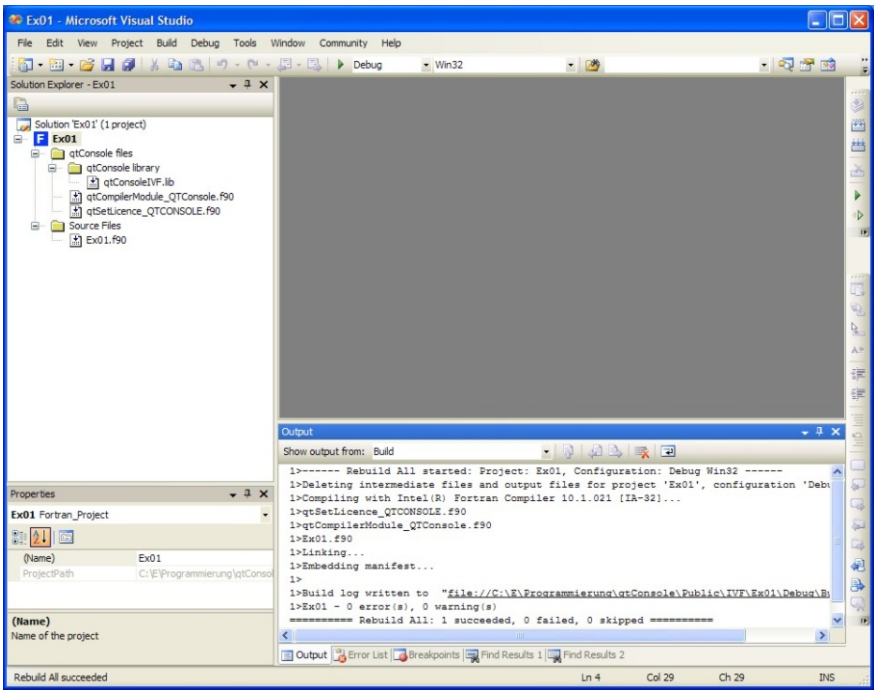

Fig. 5: The Ex01 solution and project in IVF's Visual Studio.

The screen shot above shows three qtConsole files

```
qtConsoleIVF.lib
qtCompilerModule_QTCONSOLE.f90
qtSetLicence_QTCONSOLE.f90
```

being part of the project. The latter one is the dummy licence file that you have to replace by your licence file `qtSetLicence_L2847_######.f90` (with `######` to be replaced by your qtConsole licence number). These three files have to be added to any of your projects that use qtConsole functions.

Project Settings

- Fortran preprocessor: If the `qt*.mod` files are not in the project's directory, specify a `USE` path for qtConsole `MODULE`s (files `qt*.mod`). See the following screen shot for an example.

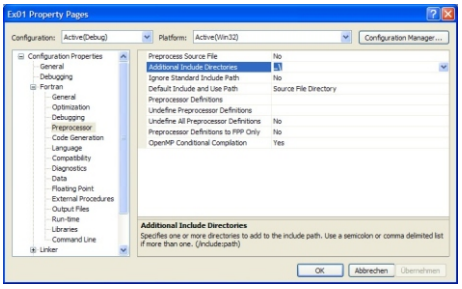

Fig. 7: Setting the USE path such that IVF finds the qt.mod files.*

- Link input:
  - Specify the Windows system library modules `user32.lib advapi32.lib`.
  - To get rid of the warning `LINK : warning LNK4098: defaultlib "LIBCMT.lib" conflicts with use of other libs; ...` which can occur in Debug mode, you may specify to ignore `LIBCMT.lib`.

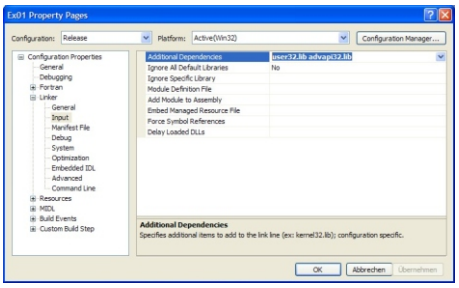

Fig. 8: Setting the link input such that the required Windows system libraries are linked.

5.4 With Silverfrost FTN95 (Win32)

The following files are supplied for usage with Silverfrost FTN95 (abbreviated FTN95) in Win32 mode.

```
QTCONSOLE.MOD
qtConsoleFTN95.lib
```

The first one is the compilation of several qtConsole `MODULE`s. The other one is the static library. Both files are found in a subdirectory named `FTN95`.
There is also a sample project file (`Ex01.ftn95p`) for usage with Plato3 in the subdirectory `FTN95\Ex01`. You may start Plato3 by loading the

project file. It contains all necessary settings to generate an executable from the example Ex01.f90. Just start the "Build" process.

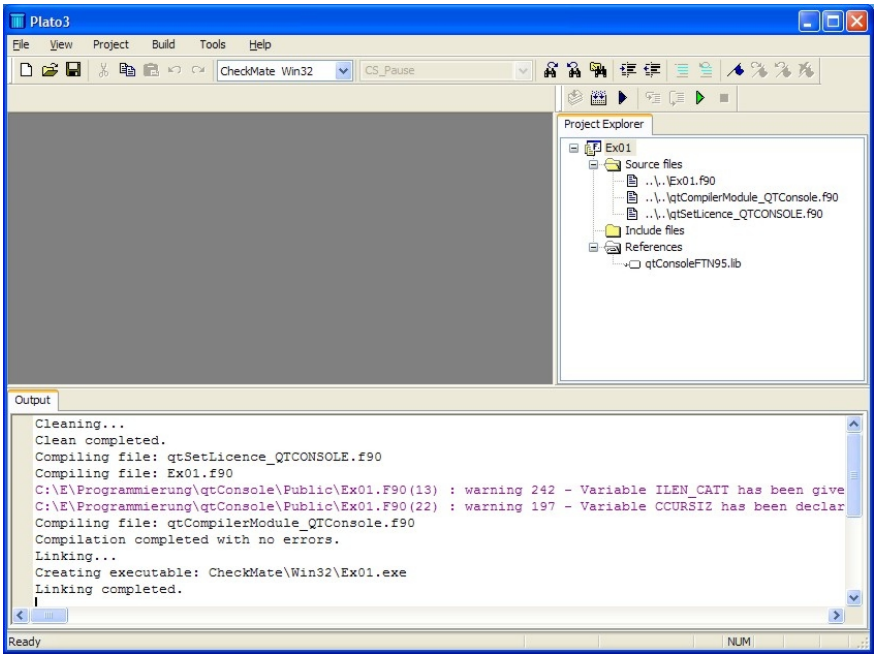

Fig. 9: The Ex01 project in Plato3.

Project Properties

FTN95 needs to find the qtConsole MODULE `QTCONSOLE.MOD`. Thus, the `MODULE path` has to be specified in the `Compiler Options | Source`. See the screen shot for an example.
Since FTN95 already links all necessary Windows system libraries there is nothing to change in linker settings for qtConsole usage.

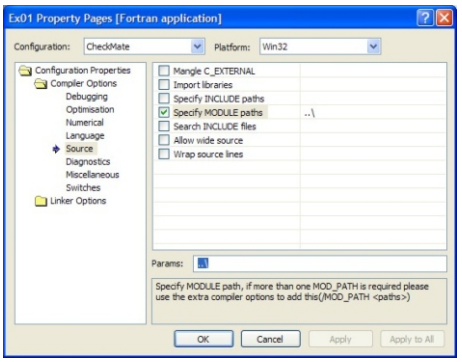

Fig. 10: Setting the MODULE path such that FTN95 finds the qtConsole.mod file.

6. System Requirements

To be able to use the qtConsole software, the following is required:

- PC with at least a Pentium processor, hard disk with about 6 MB space available.
- Supported operating systems are: Windows 2000, Windows 2003, Windows XP and compatible. Some elder Windows systems may be compatible too (depends on the Fortran compiler being used).
- Compiler systems: one of the Fortran 95 compliant compiler systems listed in the chapter “Compile & Link”.

7. Licence Agreement - Legal Conditions to use the qtConsole Software

§1 Property and Ownership

The software described in this document - called qtConsole software in the following, mainly consists of the files named qtConsole*.lib (* is a wildcard to be replaced by a compiler abbreviation as described in this document) and those mentioned in the chapter "Compile & Link". In particular these are library files ending on the .lib, pre-compiled MODULE files (ending on .mod), this documentation, and any other file mentioned in this document whose name start with the letters "qt". All these files are property of the author Jörg Kuthe. The author is represented by QT software GmbH, Germany, called QT followingly, which is authorized to allocate use rights to qtConsole. The copyright at the qtConsole software and this documentation remains with the author.

§2 Licensee and Licence File

The licensee is the user of the qtConsole software, which he has obtained from QT in the course of the purchase of a qtConsole licence. The licence file (in the file `qtSetLicence_L2847_######.f90` with `######` to be replaced by the licensee's licence number) contain identifying data of the licensee.

§3 Licence and Passing on of Components of the qtConsole Software

The licence consists of the right of the licensee to use the qtConsole software for the development of programs, i.e. thus to create executable files whose name ends on .exe. The executable file created with qtConsole may neither have the function of a compiler nor that one of a linker. It is not allowed to pass the licence file to anybody else.

§4 Transfer and Assignment of the Licence

The licensee cannot be represented by another person. This excludes particularly the rental of the qtConsole software. The licensee may sell the use licence if he reports the disposal or assignment of the licence to another licensee to QT in writing. The assignment must include the confirmation, that the selling licensee gives up his rights at the qtConsole software. The new licensee must agree to these licence conditions in writing and deliver to QT those data which are necessary for the preparation of a new licence file.

§5 Warranty

QT ensures the function ability of the qtConsole software for the period of 2 years after acquisition of the licence. In the fault case it is up to QT either to refund the paid price to the licensee or to fix the reported error in the qtConsole software. If the money paid for qtConsole is refunded, the licensee looses the right to use the software qtConsole. Complaints or faults have to be verified by the licensee by a program he provides.

§6 Use Risk and Limitations of Liability

The licensee uses the software qtConsole on risk of his own. QT is liable in maximum amount of the paid price for the qtConsole software.

§7 Declaration of Consent

The licensee gives his agreement to these conditions by the acquisition of the qtConsole licence.

8. Other Notes

The author and QT software GmbH acknowledge the rights of the owners at the brand names, trademarks and products named in this document:
Excel is a trademark of Microsoft Corporation, U.S.A..
Windows is a trademark of Microsoft Corporation, U.S.A..
“Compaq Visual Fortran” is a product of Hewlett-Packard Company, U.S.A..
“Intel Visual Fortran” is a product of Intel Corporation, U.S.A..
“Lahey/Fujitsu Fortran 95 for Windows” is a product of Lahey Computer Systems, Inc., U.S.A..
“ProFortran for Windows” is a product of Absoft Corporation, U.S.A..
“Silverfrost FTN95” is a product of Silverfrost Ltd., U.K..

Index

M

N

P

Q

R

S

T

U

V

W