

NC-Toolkit

DLL-Interface for WINDOWS

Software Development Kit

- For Multiple Axes Motor Control
- Based on your programming language
- Includes I/O-Access and "Real Time"
- under WINDOWS 98, NT 4.0, 2000 and XP

For the Version Nr. of the driver and DLL-Interface see File Version.txt

New Features are written in this color.

This Manual, along with the Software belonging to it, is protected by copyright. Except making in house backup copies any reproduction or resale is prohibited.

Copyright © 2002 by TRIMETA software GmbH, F. Stütz, D-75233 Tiefenbronn, GERMANY

Contents of the Disks

See file Readme.txt for the shipment contents.

An Important part of the documentation is EDITASC.DOC, the manual of EdiTasc.

Since EdiTasc uses the same DLL interface as NC-Toolkit, it is quite helpful as environment for testing your **MtExec** calls.

The software samples for Visual Basic 6.0 are located in directory **Vb6**, those for Visual C++ 6.0 in directory **Vc6**. There are **Hello** projects for both. The Project in **Vb6\EdiTascEngl** is a simplified Version of EdiTasc. **Vb6\EdiTascGer** is the same in german.

The DLLs and other files needed at runtime are located in directory **RunFl**. The device driver is in directory **Drv**. For the NT-Version some tools are included for registering the driver.

Important Files for the NC-Toolkit and EdiTasc

READ_xx.TXT informs about the specials of the hardware used by your driver type (xx is.the placeholder for the driver type).

Projects for VB 6.0: *.VBP

Projects for VC 6.0: *.DSW (or *.DSP)

Files with Extension ".TS"

MTASC-Programs (see Editasc manual), that are needed at runtime:

SYSTEM.TS includes all MTASC definitions needed for EdiTasc.

AUTOEXEC.TS e.g. is usually called at startup.

HELLO.TS is a MTASC program called by the HELLO projects.

PARAMSET.TS:

Contains parameter settings like standard speeds and graphic view scale.

REFSRCH.TS:

Searches the reference positions for all axes, usually on request of the user.

Files with Extension ".T"

MTASC-Programs for or written by the user:

DEMO1.T is an Example.

DRV_xx.INI:

Configuration of the driver, xx is the placeholder for the driver type communicating with the hardware.

MTASC.LIB: Import library for MTASC.DLL

MTASCDLL.H: corresponding C-header file

Access to the graphic library is included „as is“, without guaranteed success in using it. You find some hints in the EdiTasc sample project:

G3GRAPH.LIB: Import library for G3GRAPH.DLL

G3GRAPH.H: corresponding C header file

The DLLs:

MTASC.DLL: MTASC interpreter

G3GRAPH.DLL: Graphic interface, it must be in the same directory as MTASC.DLL at runtime.

CBNDLL.DLL: Needed for the software access key (dongle).

The Driver:

MTDRVX.VXD: VxD driver including the real time kernel (for Windows 98 only).

CBN.VXD: VxD driver needed for the software access key (dongle) (for Windows 98).

MTDRV.SYS: Device driver including the real time kernel (for Windows NT and 98).

MARXDEVx.SYS:

Device driver needed for the software access key (dongle) (for Windows NT).

The Low Cost Version only for Positioning

In this version the interpolating vector commands in MTASC like ML, MF, MC are not supported. Instead you have to use MP with same syntax as ML.

Installation

Install **EdiTasc** first by calling Setup.exe on disk 1. This also installs the drivers needed by NC-Toolkit. Then copy the **NC-Toolkit** disk into any (other) directory of your choice, including all subdirectories.

The sample projects – of course – require the corresponding programming language. Their EXE-files however should run in any case (as soon as EdiTasc is installed).

Copy the files in directory **RunFI** to the directory where the EXE file of your project will run. This is also required for the sample projects. For the C++ projects this is the directory **Debug** or **Release**

After restarting windows EdiTasc and the Hello samples should run.

Explore EdiTasc first to get a feeling for the possibilities of the NC-Toolkit! As mentioned, the Project in **Vb6\EdiTasc** is a simplified Version of EdiTasc.

How to install and register the driver

For Windows 98:

Since Version 6.12 SYSTEM.INI does **not** require an entry anymore. If you have installed an older version on your system, you **must remove** this entry or set a ';' in front of the device entry:

```
[ 386Enh ]
```

```
.....
```

```
;device=C:\Progra~1\EdiTasc\Mtdrv.vxd
```

Since Version 6.2x the driver for Win 98 has 2 components (Mtdrvx.vxd und Mtdrv.sys), for Win NT; however, it only has one (Mtdrv.sys, different from the 98 component).

Setup copies the Device driver **Mtdrvx.vxd** into the **System** directory (for Win 98) or **Mtdrv.sys** into **System32\drivers** (for Win NT and 98) and adds entries in the registry. This is done by the batch file REGDRV.BAT in the **Driver** directory of EdiTasc. The drivers for the software keys are installed as well. In case you want to repeat this process, just call or double click the batch file. If you are working with different driver types or versions: When switching copy the driver to be used into the appropriate System directory and reboot.

The Interface to control Axes and Hardware: MTASC.DLL

Most commands are passed via the script language **MTASC** as string. MTASC is similar to C and Basic and is described in the EdiTasc manual.

How to get all stuff initialized, it is a good idea to start up with the Hello projects.

If You want a graphic window to show the linear movement of X and Y, use the MTASC system variables **_ncMode** and **_grMode** to enable NC mode (moving the axes) or graphic mode (showing the movement) or both.

Be aware, that these variables are defined only after initializing all system variables with the MTASC function **ConfigSVar**. The required argument is a String containing the letters you want for addressing the axes. The String may be empty, yielding the default: "xyzabc".

For calling this function see MtExec lateron and the EdiTasc manual.

For the declarations of all exported functions see C header file MTASCDLL.H.

MtInit: Initializing

This function must always be called first.

Parameter (C Syntax):

HWND **hWnd**: The window handle of your main window. Messages of the DLL (errors and the MsgBox function) will appear modal to it.

LPSTR **appPath**: Path to the current main program (EXE file) and the drivers INI file.

short **nAx**: Maximum number of axes.

short **lang**: Determines the **language for error messages**.
 Ascii-Code for 'G': German, 'E': English.

ULONG **MsgTextMaxLen**: Size of the text buffer for error messages in bytes. If this parameter is 0, the default value **1000** is used.

This text buffer is accessed by the error handler **_ErrorHandler**, which is now defined as MTASC macro. Its default definition is:

```
(MacroDef "_ErrorHandler") =
  "MsgBox ThreadMsgText \"MTASC message\" MB_OK;";
```

If you prefer a different error handler (e.g. without generating a message box), you can redefine this macro.

MtInitG: Initializing the graphic interface

If you do not use graphics, ignore this function.

hDcV and **hDcR** must be the device context of the window. Why **two** device contexts? This is due to a (possibly undocumented) behaviour of Visual Basic: You will probably prefer to use windows with AutoRedraw property set True. But automatic redraw won't work unless you draw into two devices, one probably being a bitmap. G3GRAPH will do this for you if you pass the correct context handles. See the EdiTasc project of NC-Toolkit for a way how to get those two handles.

MtInitG can be called repeatedly, e.g. if the device context handles changed.

Parameter (C Syntax):

HANDLE ***hg3**: Must contain zero when called first (not a NULL pointer!). Here a handle for subsequent direct calls to G3GRAPH.DLL is returned. You will normally not need it. For further calls to MtInit pass a pointer to the value returned at the first time or 1 (not zero). This prevents allocation of a new handle.

HANDLE ***hg3rkor**: Like hg3, but for the output of paths displaced by cutter radius compensation. This is an extra option (RKOR Option). Pass 1 if you don't need a handle.

HDC **hDcV**: Device context for graphic output into the visible window.

HDC **hDcR**: Device context for graphic output into the bitmap for automatic redraw.

MtExec: Execute String as MTASC Command

This is the most important one of all functions. It expects a string as argument, which may contain any legal MTASC command that is to be executed. This is much like executing a text line from within the text editor of EdiTasc by pressing Ctrl+Enter. See EdiTasc manual for details.

A call to **MtExec** may not return immediately in one of these cases:

- The command string contains a **WAIT** Assignment. The call will return after the time has elapsed.
- More than **50 vector commands** are being sent to the driver (e.g. **FCALL** passes execution to a file containing MTASC codes), meaning that the drivers command buffer (a FiFo type buffer) gets filled up. **MtExec** continues as soon as possible.
- There is an endless loop in the code. In this situation you can invoke a break by a reentrant MtExec call to MRESET (or by changing variables that are responsible for the loop).

As long as MtExec "hangs", the program should still react to user commands, since MTASC.DLL allows reentrance. There is a limit for the number of times reentrance may occur. This is controlled by the MTASC system variable **_RL** (Reentrance Limit). It should not be 0 because endless loops will then hang up your program. The name of this variable "_RL" is not

fixed but is set by a call to the MTASC function **ConfigReent**. It is called by using MtExec and should be part of standard initialization code (see the Hello projects).

You can break and abort any pending MtExec call by

MtExec "MRESET 1;" or

MtExec "MRESET ""Your Message"";" (string as argument of MRESET in Visual Basic).

If the argument string is empty, you avoid a message when aborting execution. Otherwise, the filename and line number will be displayed in case it was a file (called by FCALL).

Sometimes it may be necessary to pass short commands that **may not be interrupted** by other events like key or mouse messages. In this case place '@' at the beginning of the string. This is advisable for MtExec calls in background routines triggered by a timer.

MtGetStat: Get Status information

used for permanent reading of current position (Coordinates) and other status information. See the EdiTasc Project for an example how to call this function. Recommended for formatted output of the coordinates into a string. For reading the position as a numeric value **MtGetData** is the better choice.

Parameter (C Syntax):

LPSTR **coord** (Coordinate Display): Pass an empty string. Only provided for compatibility with earlier versions.

LPSTR **fmt** (Format String for coord): Pass an empty string. Only provided for compatibility with earlier versions.

LPSTR **cmd** (MTASC Command): If the 1. Byte is not Blank (0x20), this String is executed as MTASC command. Ist result (the value in the last expression) must be an INTEGER (in terms of MTASC) which is returned in **ret**. Can be used for permanent watching of I/O conditions. Example: Return 1 if bit 1 or bit 2 at the printer port is set, otherwise 0:

```
cmd = "err = 0; if(0x06 != INP 0x378 0x06){err = 1;} err;"
```

short ***ret** (Result of cmd): See above.

MtStatus ***stat**: Pass NULL pointer if not used, otherwise a pointer to this structure:

```
struct MtStatus // siehe MTASCDLL.H
{
    short dcFlag; // Bit PRT_Oc and/or PRT_Mc may be set
    DcPoint dcOc, dcMc;
    struct
    {
        short mtrans : 1;
    };
    short drvFiFoLevel;
    USHORT lim; // drvh.McLimFlag
    USHORT pErr; // drvl.PosErrFlag
    USHORT drvFlag; // drvl.Flag
    DWORD vciCurr;
    short stop;
};
```

The returned values are:

mtrans: set, if any transformation is active (see MTASC function MTRANS).

drvFifoLevel: Number of elements currently used in the vector command buffer, (maximum is 50). This value can be used to find out if axes are still moving.

lim, Bit 0-7: Bitmask, tells which Axes exceed the range limits (software limitation). (Bit 0 = X etc., up to 8 Axes).

lim, Bit 8-15: Bitmask, tells which Axes exceed the range limits **in negative direction** (software limitation). (Bit 0 = X etc., up to 8 Axes).

pErr, Bit 0-7: Bitmask, tells which axes were responsible for STOPPING due to an excessive Position Error.

drvFlag, Bit 0-7: Bitmask, tells in which axes an internal driver error occurred.

vciCurr: Not documented in the standard version. If you buy the Visual Basic source code of EdiTasc its use is obvious.

stop: Set if motion was stopped (see MTASC function MSTOP).

Return Value:

1 if successful, 0 in case of an error (string length of sf or lim not OK).

MtManStep: Jogging the axes manually

Recommended for jogging by the keyboard.

Parameter (C Syntax):

short **aIndex:** Index of the axis to be moved. 0 = X-axis, 1 = Y etc.

short **dir:** Direction. +1 = positive, -1 = negative.

Return value: 1 if successful, otherwise 0.

When called once the function moves the indicated axis at steps defined in DRV_xx.INI, entry **manVmin** (see EdiTasc manual, Appendix).

Repeated calls (about at auto repeat rate of the keyboard) cause an increasing speed. The maximum is defined by **manVmax**. In EdiTasc jogging this way is invoked by AltGr + Cursor keys.

To make sure the axes won't keep on running until the end of the step, call

MtManStep(-1, 0)

which should be called in the KeyUp event. Don't call this when the keys were only pressed once or twice. See **MvManCnt** in the Visual Basic project EdiTasc.

In case any axis exceeds its software limits (siehe MTASC command **MLIMIT** and MtGetStat) MtManStep allows jogging back into the legal area. This is the only chance to move in this situation. The ML, MF functions do not work then, unless you disable limit watching by **MLIMIT '0'**;

MtGetData: Read values of MTASC variables

This function expects – like MtExec – an MTASC command as String. It may have any number of expressions. The result of the last expression is returned.

Parameter (C syntax):

LPCSTR **cmd**: String with MTASC command. The result of the last expression is returned in the corresponding pointer lpi, lpl, lpdbl oder str. It depends on the data type of the resulting value.

short ***lpi**: Receives data type WORD.

long ***lpl**: Receives data type DWORD. (New type with 32 Bit, may not be mentioned in the EdiTasc manual).

double ***lpdbl**: Receives data type DOUBLE.

LPSTR **str**: Receives data type CHAR oder CHAR-Array (STRING). A CHAR is placed at the beginning of str. The length of str is defined by its terminating 0. On entry str may not contain any other 0. Be careful when using a static string as container, since the returned string is copied including its terminating 0.

Return value:

An Integer telling you the data type:

0: Data type not supported (undefined or Arrays other than CHAR)

1: WORD

2: DWORD

3: DOUBLE

4: CHAR

5: CHAR-Array (STRING)

Example for C:

```
short valS;
long valL;
double valDbl;
char valStr[] = "          ";
....
MtGetData("stat = \"LOW\"; if(inp 0x378 1){stat = \"HIGH\";} stat;",
          &valS, &valL, &valDbl, valStr);
```

Returns String "LOW" bzw. "HIGH" in in valStr, depending on the lowest bit of the printer port.

In the Visual Basic project EdiTasc you find a useful function called **MtGet**, which allows convenient reading of all data types. It returns the data as data type Variant.

In C you could create similar functions like **MtGetStr** or **MtGetDbl** etc.

The Sample-Projects

The MTASC program file HELLO.TS is called by all HELLO projects. It calls in turn AUTOEXEC.TS and SYSTEM.TS and sets some MTASC variables. Although AUTOEXEC.TS and SYSTEM.TS include much more than what is actually needed by HELLO itself, this compatibility to EdiTasc makes sense, because HELLO.EXE is able to run by just copying it into the EdiTasc directory together with HELLO.TS.

Project HELLO in "Vb6\Hello" for Visual Basic 6.0

This sample is reduced to its minimum. It shows, how to move the axes with minimal expence.

Pay attention to the use of the **RunFlag** variable. It is used to find out if an MtExec-Command is still running. In this case **it is not allowed to leave the program!**

Project EdiTasc in "Vb6\Editasc" for Visual Basic 6.0

Here most DLL functions can be studied in action.

Jogging is possible by the cursor keys together with the AltGr-key.

Click in the menue **File** on **Choose**, in the File Dialog Box you choose DEMO1.T. Show the program as graphic either with **File / Graphic (Simulation)** or with **F4**. The execution is started by **F5**.

With **File / Edit** a simple text editor opens showing the text of the program. Here you can execute single lines with Ctrl+Enter as MTASC commands (see EdiTasc manual).

Project HELLO in "Vc6\Hello" for Visual C++ 6.0

This looks like the Visual Basic sample project HELLO. It is structured as a "Dialogbased" **MFC-Application** (Workspace Hello.dsw). The strukture can be easily recognized in the development-system.

The program **Hello.exe** can be compiled with Microsoft Visual C++ 6.0.

As above, pay attention to the use of the **RunFlag** variable. It is used to find out if an MtExec command is still running. In this case **it is not allowed to leave the program.**

., which again calls AUTOEXEC.TS and SYSTEM.TS and sets some MTASC variables.

Although AUTOEXEC.TS and SYSTEM.TS include much more than is actually needed ba HELLO itself, this compatibility to EdiTasc makes sense, because HELLO.EXE is able to run by just copying it to the EdiTasc directory.

Project HELLO in "Vc6\HelloTest" für Visual C++ 6.0

Is extended by a coordinate und status display.