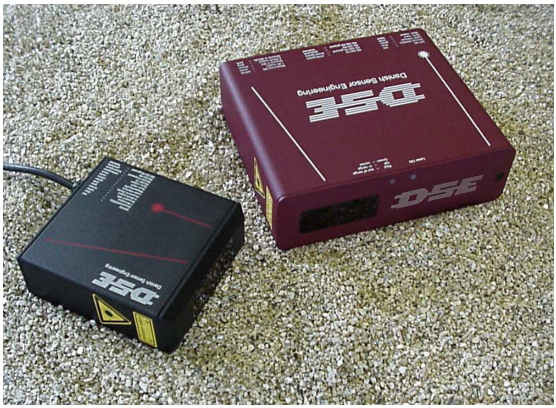
ODS Sensor API

Programmers Manual

DSE.dll Version 4

(Universal ODS Select-X Single / Master Sensor Support)



Danish Sensor Engineering

Columbusvej 3 · DK-2860 Søborg · Denmark

 DSE - Danish Sensor Engineering ApS
Columbusvej 3
DK-2860 Søborg
Denmark

Tel: (+45) 39 66 71 44
Fax: (+45) 39 66 71 45
e-mail: dse@sensor.dk

6TH edition, November, 2014

Printed at DSE ApS

TABLE OF CONTENTS

DESCRIPTION

The ODS Sensor API is designed to make it easy for the application programmer to obtain measurement data from ODS sensors within a Windows™ application. This software, intended for handling a single sensor, supports all the ODS sensor models in the 2002 product program. Multi sensor DLL's are offered on a special order basis.

About this maual

The code snippets and examples in this manual are written in the C/C++ syntax unless otherwise stated.

Design Philosophy

The DLL has been designed to be robust in use, and it is thus allowed to call any function at any time without any risk of runtime errors. If a function is unable to perform its intended task, it will merely return DSE_INVALID (a numerical constant with the value -1) or DSE_FAILURE (with the value 0).

Similarily, after calling DSE_Open, the DLL will maintain synchronization with the sensor at all times, in order to avoid data loss and to allow quick starting and stopping of the data queue as well as making status readouts possible at all times.

To keep the DLL simple yet robust, the use of callback functions and windows messages has been avoided. Instead, a simple polling strategy has been adopted. It is left to the application programmer to choose and implement the actual polling-loop. Functions for obtaining status flags and queue size etc. have been provided as described below, and have been optimized for frequent calling.

DLL OPERATION

DLL Functionality

The ODS Sensor API can be used with any development environment (eg. programming language) capable of calling functions residing in a DLL (Dynamic Link Library). While DSE provides the "#include" files for a limited number of popular languages, it will be necessary for the developer to make these manually for other languages. Se also Apendix A. The files provided ought to ease this process, serving as a template.

We recommend using static linking with DSE.dll, but loading the DLL at runtime (dynamic linkage) should be quite possible (but has not been tested, nor is it currently supported by DSE).

Status Flags

The DLL maintains several status flags to facilitate detection of various error conditions, these flags can be read as a bit-coded integer by calling DSE_Status(). Please refer to the reference section for further details.

Initialization

After calling DSE_Open() with appropriate parameters, the DLL will open the chosen serial port, allocate a measurement queue, and try to synchronize with the connected sensor.

After initializing the DLL, the most recently converted measurement value can be obtained by calling DSE_PeekValue(). Note that since this function does not make use of the internal queue, there is no way to ensure that a given measurement hasn't already been read, nor is it possible to know if one or more measurements has been lost between subsequent calls to this function.

Queue Functions

The internal queue will hold at least 2 seconds of data to prevent loss of measurements while the application is processing data or is otherwise busy.

Calling DSE_StartQueue() will make the DLL store the received measurements in the internal queue, from which the measurements can be read in a first-in-first-out (FIFO) manner. This is done by calling DSE_PeekQueue() or DSE_PopQueue(). The former doesn't change the queue while the latter will remove the measurement from the queue. Thus, if measurements are used in more than one place, one would use DSE_PeekQueue() for all readouts except the last, where DSE_PopQueue() would be used in order to remove the value, ensuring subsequent calls to DSE_PeekQueue() will return the next measurement in the queue.

To check whether any measurements are waiting in the queue, the current queue size can be obtained by calling DSE_QueueSize(), which will return the number of measurements in the queue as an integer. If the number returned is greater than zero, measurements are waiting in the queue.

The data gathering can be stopped at any time by calling DSE_StopQueue(). Measurements already present in the queue will remain available for readout by DSE_PeekQueue and/or DSE_PopQueue, but no further data will be stored in the queue. The queue can be emptied of all measurements at any time, using the DSE_ClearQueue() function.

Deinitialization

When the application is finished using the DSE.dll, it can be closed by calling DSE_Close(), causing the serial port to be released as well as deallocating the queue.

SELECT Functionality

In order to allow re-programming of SELECT-X enabled sensors, a number of SELECT-X support functions have been implemented. Please refer to the sensor documentation and datasheets for further information about SELECT-X functionality and availability. Note that the functions used for SELECT-X programming differs slightly from those used for SELECT-2 programming (as used with older ODS sensor models).

To use these functions sucessfully, a SELECT-X capable sensor should be attached and powered up, and the apropriate Com port should be opened using a call to DSE_Open().

To obtain the current SELECT-X settings for the current sensor, first call DSESEL_Check() to download the settings from the sensor. If the download went well, the function will return DSE_SUCCESS (1), if not DSE_FAILURE (0) will be returned.

When the above function succeeds, call DSESEL_Get Bit(BitMask) and/or DSESEL_GetPar(ParNum) to obtain the downloaded SELECT-X settings. using an apropriate bit-mask or word parameter as argument (please refer to the reference section for valid masks and parameters and their symbolic constant names).

To program (change) the SELECT-X settings stored in the currently opened sensor, first call DSESEL_SetBit(BitMask, BitVal) and/or DSESEL_SetPar(ParNum, ParVal) as necessary, where BitMask/ParNum is the mask/number (or its symbolic constant) for the bit or parameter that is to be changed, and BitVal/ParVal is the new value for the bit or paramter (true/false for bits, integer number for parameters). Repeat the above procedure as necessary for each bit and/or parameter that needs changing. Finally perform the actual SELECT-X programming by calling DSESEL_Select().

After successful programming, the DSESEL_Select() function will return DSE_SUCCESS (1) otherwise DSE_FAILURE (0) is returned. If successful, DSESEL_GetBit(BitMask) and DSESEL_GetPar(ParNum) can be used to verify the written values. No additional call to DSESEL_Check() is needed after programming with DSESEL_Select(), as this function is called internally by DSESEL_Select() to verify correct programming.

Detect Functionality

To facilitate easy and flexible operation of programs using the DLL, additional functions have been implemented to enable auto-detection of the sensor and its settings (COM port, baud rate & data telegram size).

NOTE: The auto-detect functions (prefixed with "DSEDET_") assume that the DSE.dll is NOT initialized (eg. Not open), and will return DSE_INVALID (-1) if this is the case. Furthermore, please note that during auto-detection, the queue and the flags may change. After detection the DLL is left closed, thus allowing a call to DSE_Open() with the newly detected settings.

Auto-detection starts with the determination of available COM ports. This is done by calling DSEDET_DetectPorts(), which will return the number of ports found.

The ports can then (if necessary) be obtained by calling DSEDET_GetPort(), which when given a number between zero and the number returned by DSEDET_DetectPorts() MINUS ONE, will return the actual COM port number for that port.

After calling DSEDET_DetectPorts(), call DSEDET_DetectSensors() to enumerate the sensor(s) on the (detected) ports. The function will return the number of sensors found.

To obtain the settings (COM port, baud rate and number of telegram bits) for a detected sensor, call DSEDET_GetSensorPort(), DSEDET_GetSensorRate() and DSEDET_GetSensorBits(), giving the sensor number as argument to each function. This argument must lie between zero and the number returned by DSEDET_DetectSensors() MINUS ONE.

The values returned by DSEDET_GetPort() (but NOT DSE_GetSensorPort()), DSEDET_GetSensorRate() and DSEDET_GetSensorBits() are designed to be used as arguments for DSE_Open().

NOTE: The value returned by DSEDET_GetSensorPort() is an index, which should be used as argument to DSEDET_GetPort() to obtain the actual COM port number! Accordingly the following syntax can be used to obtain the COM port for a detected sensor in one operation:

```
Port=DSEDET_GetPort(DSEDET_GetSensorPort(SensNo));
```

While the two other arguments can be obtained directly, like this

```
Rate=DSEDET_GetSensorRate(SensNo);
Bits=DSEDET_GetSensorBits(SensNo);
```

To open a detected sensor directly, use this syntax:

```
DSE_Open
(
   DSE_GetPort(DSE_GetSensorPort(SensNo)),
   DSE_GetSensorRate(SensNo),
   DSE_GetSensorBits(SensNo)
);
```

Note: "SensNo" is assumed to be an integer variable holding the index of the detected sensor to be opened (typically 0, indicating the first/only sensor detected).

Compatibility

The DSE.dll has been tested with Borland C++ Builder 5.0 and Delphi 4.0, but should work with any Win32 programming platform with support for calling functions in a DLL. The DSE.dll file must reside in the Windows directory, the system directory, a directory on the path, or in the application directory itself at runtime, in order for an application using the DSE.dll to work. It is recommended to place the DSE.dll in the application directory.

Language Compatibility

Specific to C++ Builder it is necessary to have the “DSE.lib” file available on the path or in the project directory during development (this may also be the case for other C/C++ compilers). Also, the DSE.h file must be included (using the #include statement) in the source file(s) that will be calling the functions in the DLL.

Similarly, the “DSE.inc” file must be included (with the {&include} statement) in the relevant source files when working with Delphi or other Pascal compilers (alternatively, the contents of this file can be pasted into the source files using the DLL functions).

Upon request, DSE may be able to supply a "DSE.bas" file, which can be used with VisualBasic, in much the samme maner as "DSE.inc" is used with Pascal. This file however, is largely untested.

REFERENCE

Standard Functions

These functions, prefixed with “DSE_”, are used for normal operation of the DLL, including opening and closing ports, managing the queue and performing measurement and status readouts.

int DSE_Open(int Port, int Rate, int Bits)

Initializes and opens the serial port indicated by the Port argument, at the baud rate indicated by Rate argument and using the telegram size indicated by the Bits argument. The Port argument must be a number indicating a valid serial port number (while some checking is done to ensure an available port is chosen, there is no guarantee that a sensor is present on the port). It is necessary to call this function before any measurements can be obtained from the DLL; this is typically done at program start. The function returns DSE_SUCCESS (1) if the port was opened, DSE_INVALID if invalid arguments was given and DSE_FAILURE (0) if the port could not be opened successfully.

Valid argument values:
Port – Any valid COM port number not already in use.
Rate – DSE_RATE_38400 or DSE_RATE_115200
Bits – DSE_BITS_16 or DSE_BITS_18

Flags affected:
DSE_CLOSED – Flag is cleared.
(Other flags may be reset to default values).

int DSE_OpenTcp(char* Host, int Port)

Initializes and opens a TCP/IP connection to the network-host indicated by the Host argument, using the TCP/IP port indicated by the Port argument. An Ethernet-capable Sensor or supported Serial-to-Ethernet converter must be present on the network at the given host and port number. The function allocates the various internal data structures and then initiates data synchronization. The Host Argument must be a C-string (null-terminated byte-array of ASCII characters) containing either a valid IP address (“aaa.bbb.ccc.ddd”) or a valid hostname (“host[.sub-domains].domain”). The Port

argument must be a number indicating a valid TCP/IP port. It is necessary to call this function before any measurements can be obtained from the DLL (via Ethernet, please refer to the DSE_Open function for serial connections). Returns are DSE_SUCCESS (1) if the port was opened, DSE_FAILURE (0) otherwise.

Valid argument values:
Host - Any valid string containing an IP address or a hostname.
Port – Any valid TCP/IP port number (1-65535).

Flags affected:
DSE_CLOSED – Flag is cleared.

int DSE_Close(void)

Closes the serial port and deallocates dynamic structures. After calling DSE_Close() synchronization with the sensor is lost and no further data will be processed. Furthermore, any data present in the queue will be lost at this point. Returns DSE_SUCCESS (1) if the port was closed, DSE_FAILURE (0) if the port was already closed or an error occured.

Flags affected:
DSE_CLOSED – Flag is set.
DSE_STARTED – Flag is cleared.
(Other flags may be reset to default values).

int DSE_StartQueue(void)

Instructs the DLL to gather subsequent measurements in its queue, this continues until the queue overflows or DSE_StopQueue() or DSE_Close() is called. In case of overflow, measurement gathering is suspended, NOT stopped, and will resume automatically if or when the overflow condition is resolved (typically when popping one or more measurements from the queue), there is no need to call DSE_StartQueue() again in this case. Returns DSE_SUCCESS (1) unless the DLL is closed, in which case DSE_FAILURE (0) is returned.

Flags affected:
DSE_STARTED – Flag is set.

int DSE_StopQueue(void)

Stops measurement gathering. The DLL will continue to maintain synchronization with the sensor, and while measurements may be lost, it is still possible to obtain the latest valid measurement received by calling DSE_PeekValue(). Also the queue will remain valid, so any measurements stored in it up to this point will be available for readout (using DSE_PeekQueue() and/or DSE_PopQueue()). Measurement gathering can be resumed at any time by calling DSE_StartQueue(). Returns DSE_SUCCESS (1) unless the DLL is closed, in whitch case DSE_FAILURE (0) is returned.

Flags affected:
DSE_STARTED – Flag is cleared.

int DSE_PopQueue(void)

Returns (and removes) the front (oldest) measurement in the queue. To avoid removing the measurement from the queue, use DSE_PeekQueue() instead. Returns DSE_INVALID (-1) if the queue is empty or if the DLL is closed.

Flags affected:
DSE_OVERFLOW – Flag is cleared.

int DSE_PeekQueue(void)

Returns (without removing) the front (oldest) measurement in the queue. Returns DSE_INVALID (-1) if the queue is empty or if the DLL is closed.

Flags affected:
None.

int DSE_PeekValue(void)

Returns the last valid measurement processed by the DLL, bypassing the queue (the value will still be queued, if the queue is running). Returns DSE_INVALID (-1) if no valid measurement has been obtained or if the DLL is closed.

Flags affected:
None.

int DSE_QueueSize(void)

Returns the number of measurements currently in the queue. Returns DSE_INVALID (-1) if the DLL is closed.

Flags affected:
None.

int DSE_ClearQueue(void)

Removes all values from the queue. Returns DSE_FAILURE (0) if the DLL is closed, otherwise DSE_SUCCESS (1) is returned.

Flags affected:
DSE_OVERFLOW – Flag is cleared.

int DSE_Status(bool Clear=false)

Returns the status flags, combined (bit-coded) into a single integer value. This function can be called even when the DLL is closed. Note that the Clear argument is optional and will (if not supplied) default to false. If DSE_Status() is called with true as the argument, the persistent flag DSE_SYNCERROR will be cleared afterwards (See the “Status Flags” section for further information).

Valid argument values:
Clear – True or False (defaults to False if ommited).

Flags affected:
DSE_SYNCERROR – Flag is cleared if true is given as argument.

SELECT Functions

These functions, prefixed with “DSESEL_”, are used when checking or performing SELECT-X programming. They will only work with SELECT-X enabled sensors. Furthermore, for SELECT-X functions to work, it is required that the DLL has been initialized (by a previous call to DSE_Open).

NOTE: Some of these functions may affect and/or clear the queue as well as affect various flags. The flags should be ignored during programming, and unread data should not be left in the queue.

int DSESEL_Check(void)

Reads the current SELECT-X settings of the sensor, allowing subsequent calls to DSESEL_GetBit(BitMask), DSESEL_GetPar (ParNum). The function returns DSE_SUCCESS (1) if the settings were successfully obtained from the sensor and DSE_FAILURE (0) otherwise (typically if the sensor is not in programming mode).

Flags affected:
Various (should be ignored).

int DSESEL_GetBit(int BitMask)

Returns the current setting of the indicated SELECT-X bit-flag parameter. Only after a call to DSESEL_Check() or DSESEL_Select() will this function return a meaningful value, otherwise DSE_INVALID (-1) will be returned.

Valid argument values:
BitMask – DSE_BIT_EXTMODE (0), DSE_BIT_LVLMODE (1) , DSE_BIT_ERRMODE (2) , DSE_BIT_SAMMODE (3) , DSE_BIT_DIFMODE (4).

Possible return values:
true (1), false (0) or DSE_INVALID (-1) – Corresponding to the setting of the SELECT-X bit-flag indicated by the BitMask argument, or DSE_INVALID in case no SELECT-X settings have been previously read from the sensor or if the argument is out of range.

Flags affected:
None.

int DSESEL_GetPar (int ParNum)

Returns the current setting of the indicated SELECT-X word parameter. Only after a call to DSESEL_Check() or DSESEL_Select() will this function return a meaningful value, otherwise DSE_INVALID (-1) will be returned.

Valid argument values:
ParNum – DSE_PAR_GRPSIZE (1), DSE_PAR_GRPZERO (2), DSE_PAR_COMRATE (3), DSE_PAR_MEDSIZE (4).

Possible return values:
0-? (depending on the parameter being read) or DSE_INVALID (-1) – Corresponding to the setting of the SELECT-X word parameter indicated by the ParNum argument, or DSE_INVALID in case no SELECT-X settings have been previously read from the sensor or if the argument is out of range.

Flags affected:
None.

int DSESEL_SetBit(int BitMask, int Value)

Stores a new value forthe indicated SELECT-X bit-flag parameter. Only after a call to DSESEL_Select() will the stored value be written to the sensor.

Valid argument values:
BitMask – DSE_BIT_EXTMODE (0), DSE_BIT_LVLMODE (1) , DSE_BIT_ERRMODE (2) , DSE_BIT_SAMMODE (3) , DSE_BIT_DIFMODE (4) , DSE_BIT_RECMODE (5).
Value – true (1) or false (0).

Possible return values:
DSE_SUCCESS, DSE_FAILURE or DSE_INVALID (-1) – If the value was correctly stored, DSE_SUCCESS is returned, otherwise DSE_FAILURE. DSE_INVALID is returned if the parameter or value arguments are out of range.

Flags affected:
None.

int DSESEL_SetPar (int ParNum, int Value)

Stores a new value for the indicated SELECT-X word parameter. Only after a call to DSESEL_Select() will the stored value be written to the sensor.

Valid argument values:
ParNum – DSE_PAR_RUNSIZE (1), DSE_PAR_RUNZERO (2), DSE_PAR_COMRATE (3), DSE_PAR_MEDSIZE (4), DSE_PAR_AVGSIZE (5).
Value – 0- ? (depending on the parameter being set).

Possible return values:
DSE_SUCCESS, DSE_FAILURE or DSE_INVALID (-1) – If the value was correctly stored, DSE_SUCCESS is returned, otherwise DSE_FAILURE. DSE_INVALID is returned if the parameter or value arguments are out of range.

Flags affected:
None.

int DSESEL_Select(void)

Uploads the SELECT-X settings, previously supplied via the DSESEL_SetBit and DSESEL_SetPar to the sensor. The function returns DSE_SUCCESS (1) if the sensor were successfully programmed and verified with the new settings, and DSE_FAILURE (0) if the programming was unsuccessful or if the verification failed.

NOTE: This function will call DSESEL_Check() internally, in order to verify correct programming. Thus it is not necessary to call DSESEL_Check() manually before using DSESEL_GetBit() and DSESEL_GetPar() to obtain the values just stored in the sensor.

Valid argument values:
Mode – DSESEL_MODE_AVG (0), DSESEL_MODE_LVL (1), DSESEL_MODE_EXT (2) or DSESEL_MODE_STD (3).
Par1 – 2-200.
Par2 – 0-99.

Flags affected:
Various (should be ignored).

int DSESEL_GetSerial(void)

Downloads the Serial Number of the attached ODS sensor.

Using the Status Flags

The DSE_Status() function returns an integer containing the various status flags, in bit-coded form. In order to check the Boolean status of a given flag, it is necessary to use the following formula (shown in pseudo-code):

(STATUS_INTEGER ***AND*** FLAG_BIT_CODE) = FLAG_BIT_CODE

Where:

"STATUS_INTEGER" is either an integer variable containing the value obtained with previous a call to DSE_Status() or a direct call to DSE_Status() itself

"FLAG_BIT_CODE" is the bit-code of the Flag to be tested (see below for predefined constants that can be used here).

"***AND***" should be the binary AND (NOT the logical) operator of the language in question.

"=" should be the numerical equality operator of the language in question (for C / C++ this would be "==").

NOTE: In languages such as C or C++, where an integer will evaluate to Boolean "False" when zero and "True" when non-zero, the following, more compact formula will work as a valid, logical expression:

STATUS_INTEGER ***AND*** FLAG_BIT_CODE

The seven valid status bit-codes are defined in the header files accompanying the DLL (DSE.h, DSE.pas, and DSE.bas). This has been done to improve readability and to ensure compatability with future DLL versions. When using the DLL from languages not compatible with any of the supplied header files it is suggested that these values be defined as constants according the following scheme (please refer to the file “DSE.h” for verification of these values):

Name	Value
DSE_CLOSED	1
DSE_STARTED	2
DSE_OVERFLOW	4
DSE_NOSYNC	8
DSE_SYNCERROR	16
DSE_COMMERROR	128
DSE_COMMSTALL	256

NOTE: Although it is strongly recommended to do so, it is not madatory to use these symbolic constants. It is equally possible to use their numerical equivalence instead. However, DSE reserves the right to change these numerical values in future versions of the DLL.

NOTE: The DSE_SYNCERROR flag is special, in that it is never cleared automatically. This ensures that the flag will remain set if a transient sync. error was detected. This is as opposed to the DSE_NOSYNC flag, which will remain set only as long as the sync. error state persists. To clear this “persistent” flag, supply true as argument to the DSE_Status() function. Clearing the flag will take effect AFTER the current call to DSE_Status() function, thus allowing the flag to be read and afterwards cleared in a single call to the function.

The polarity of these flags has been chosen in such a way that, when operating nominally (eg. No error conditions and with the queue stopped), DSE_Status() will return DSE_OK (0). This has been done to facilitate easy error checking in languages where an integer will evaluate to Boolean “False” when zero and “True” when non-zero, as the following code example illustrates:

```
if(!DSE_Status())
   // …do normal operations here.
else
   // …do error processing here.
```

NOTE: Notice that a logical negation operator "!" has been placed in front of the call to DSE_Status() within the "if" statement. This was merely done for aesthetic reasons, to allow the "normal operation" to follow the "if" statement and relegate the "error processing" to below the "else" statement. The code would work just as well without the "!", providing that the "normal operation" code is swapped with the "error processing" code.

Other Predefined Constants

In addition to the seven status bit-codes, discussed in the previous section, a number of other symbolic constants are defined in the header/include files accompanying the DLL (DSE.h and DSE.pas).

NOTE: Although it is strongly recommended to do so, it is not madatory to use these symbolic constants. It is equally possible to use their numerical equivalence instead (shown in the parenthesis). However, DSE reserves the right to change these numerical values in future versions of the DLL.

DSE_RATE_38400 (38400),
DSE_RATE_115200 (115200),
DSE_RATE_ 230400 (230400),
DSE_RATE_ 460800 (460800),
DSE_RATE_ 921600 (921600)
– The allowable baud rates for the DSE_Open() function. (Please refer to the Sensor Manual for further information regarding the baud and data rates).

DSE_BITS_16 (16),
DSE_BITS_18 (18)
– The number of bits in the small respectively large data telegram size used by different sensor models. (Please refer to the Sensor Manual for further information regarding the data telegram size.)

DSE_INVALID (-1)
– The value returned whenever a function is unable return meaningful data (ie. when the DLL has not been opened).

DSE_SUCCESS (1),
DSE_FAILURE (0)
– The return values used by certain functions to indicate success or failure.

DSE_OK (0)
– The value returned by DSE_Status() when status is nominal, corresponding with all flags being clear (ie. The DLL is open and presently synchronized with a sensor, the queue is running and not overflowed and no sync. errors have been detected since last clearance of DSE_SYNCERROR).

DSE_BIT_EXTMODE (0),
DSE_BIT_LVLMODE (1),
DSE_BIT_ERRMODE (2),
DSE_BIT_SAMMODE (3),
DSE_BIT_DIFMODE (4)
– The bit-flags used during SELECT-X programming to indicate the various on/off settings. (Please refer to the Sensor Manual for further information about SELECT-X settings).

DSE_PAR_RUNSIZE (1),
DSE_PAR_RUNZERO (2),
DSE_PAR_COMRATE (3),
DSE_PAR_MEDSIZE (4),
DSE_PAR_AVGSIZE (5)
– The parameter numbers used during SELECT-X programming to indicate the various numerical settings. (Please refer to the Sensor Manual for further information about SELECT-X settings).