MscDll Reference Manual

This users manual has been optimized for an interactive web view.
Use the PDF version only, if no access to the web version is available.

Introduction

1 Introduction

1.1 Imprint

Title	Irinos MscDll reference manual
Manufacturer	Messtechnik Sachs GmbH Siechenfeldstraße 30/1 D-73614 Schorndorf Germany Phone +49 7181 99960-0 post@messtechnik-sachs.de
For use with	Measurement modules Irinos IR
Copyright note	© 2015 - 2016 Messtechnik Sachs GmbH
Trademarks	All product names used in this manual are trademarks of their respective owners.
Material-No.	785-1019
Change not	Subject to change without notice.
Release date	17/11/2016

1.2 Revision history

Version	**Datum**	**Changes**
A	2016-02-17	First revision
B	2016-	Data format for Opcode opcREv[89] corrected.

	11-02	
C	2016-11-17	New opcode opcBIORO[99] for reading the current state of the in-/outputs. Available from firmware version 1.4.x.x.

1.3 Legal notes

1.3.1 Terms of use for software & documentation

I. Protection rights and scope of use

Messtechnik Sachs provides operating instructions, manuals, documentation, and software programs - all collectively referred to as "LICENSED OBJECT" below - either on portable data storage devices (e.g. diskettes, CD ROMs, DVDs, etc.), in written (printed) form or in electronic form, for a fee and/or free of charge. The LICENSED OBJECT is subject to proprietary safeguarding provisions among other regulations. Messtechnik Sachs or third parties have protection rights for this LICENSED OBJECT. In so far as third parties have whole or partial right of access to this LICENSED OBJECT, Messtechnik Sachs has the appropriate rights of use. Messtechnik Sachs permits the user the use of the LICENSED OBJECT under the following conditions:

1.1) Scope of use for electronic documentation

a) With the acquisition/purchase or relinquishment of a LICENSED OBJECT, you as the user acquire a simple, non-transferable right of use with regard to the respective LICENSED OBJECT. This right of use authorises the user to use the LICENSED OBJECT for the user's own, exclusively company-internal purposes on any number of machines within the user's business premises. This right of use includes exclusively the right to save the LICENSED OBJECT on the central processors (machines) used at the location.

b) Irrespective of the form in which operating instructions and/or documentation are provided, the user may furthermore print out any number of copies on a printer at the user's location, providing this printout is printed with or kept in a safe place together with these complete terms and conditions of use and other user instructions.

c) With the exception of the Messtechnik Sachs logo, the user has the right to use pictures and texts from the operating instructions/documentation for creating the user's own machine and system documentation. The use of the Messtechnik Sachs logo requires written consent from Messtechnik Sachs. The user is responsible for ensuring that the pictures and texts used match the machine/system or the product.

d) Further uses are permitted within the following framework:

Copying exclusively for use within the framework of machine and system documentation from electronic documents of all documented supplier components. Demonstrating to third parties

exclusively under guarantee that no data material is stored wholly or partly in other networks or other data storage devices or can be reproduced there.
Passing on printouts to third parties not covered by the regulation in item 3, as well as any processing or other use are not permitted.

1.2) Scope of use for software products
For any type of Messtechnik Sachs software including the associated documentation, the customer shall receive a non-exclusive, non-transferable and time-unlimited right of use on a certain hardware product or on a hardware product to be determined in individual cases. Messtechnik Sachs shall remain the owner of the copyright as well as of any other industrial property rights. The customer may make copies for back-up purposes only. Any copyright notes may not be removed.

2. Copyright note
Every LICENSED OBJECT contains a copyright note. In any duplication permitted under these provisions, the corresponding copyright note of the original document concerned must be included:
Example: © 2016, Messtechnik Sachs GmbH,
D-73614 Schorndorf

3. Transferring the authorisation of use
The user can transfer the authorisation of use re. the respective LICENSED OBJECT as per these provisions in the scope and with the limitations of the conditions in accordance with items 1 and 2 completely to a third party. The third party must be made explicitly aware of these terms and conditions of use.

II. Exporting the LICENSED OBJECT
When exporting the LICENSED OBJECT or parts thereof, the user must observe the export regulations of the exporting country and those of the acquiring country.

III. Warranty
1. Messtechnik Sachs products are being further developed with regard to hardware and software. If the LICENSED OBJECT, in whatever form, is not supplied with the product, i.e. is not supplied on a data storage device as a delivery unit with the relevant product, Messtechnik Sachs does not guarantee that the electronic documentation corresponds to every hardware and software status of the product. In this case, the printed documentation from Messtechnik Sachs accompanying the product is alone decisive for ensuring that the hardware and software status of the product matches that of the electronic documentation.
2. The information contained in an item of electronic documentation can be amended by Messtechnik Sachs without prior notice and does not commit Messtechnik Sachs in any way.

3. Messtechnik Sachs guarantees that the software program it created agrees with the change description and program specification but not that the functions included in the software run entirely without interruptions and errors or that the functions included in the software can run or meet the requirements in all combinations selected by and in all conditions of use designated by the acquirer.

IV. Liability/limitations on liability

1. Messtechnik Sachs provides LICENSED OBJECTS to allow the user to use - in conformity with the contract - Messtechnik Sachs products which require software for proper operation, or to assist the user in creating the user's machine and system documentation. In the case of electronic documentation which in the form of data storage devices does not accompany a product, i.e. which is not supplied together with that product, Messtechnik Sachs does not guarantee that the electronic documentation separately available/supplied matches the product actually used by the user.
The latter applies particularly to extracts of the documents for the user's own documentation. The guarantee and liability for separately available/supplied portable data storage devices, i.e. with the exception of electronic documentation provided on the Internet/Intranet, are limited exclusively to proper duplication of the software, whereby Messtechnik Sachs guarantees that in each case the relevant portable data storage device or software contains the latest status of the documentation. In respect of the electronic documentation on the Internet/Intranet, it is not guaranteed that this has the same version status as the last printed edition.
2. Furthermore, Messtechnik Sachs cannot be held liable for the lack of economic success or for damage or claims by third parties resulting from use of the LICENSED OBJECTS by the user, with the exception of claims arising from infringement of protection rights of third parties concerning the use of the LICENSED OBJECTS.
3. The limitations on liability as per paragraphs 1 and 2 do not apply if, in cases of intent or wanton negligence or lack of warranted quality, liability is absolutely necessary. In such a case, the liability of Messtechnik Sachs is limited to the damage recognisable by Messtechnik Sachs when the specific circumstances are made known.

V. Safety guidelines/documentation

Guarantee and liability claims in conformity with the regulations mentioned above (items III and IV) can only be made if the user has observed the safety guidelines of the documentation in conjunction with use of the machine and its safety guidelines or the terms and conditions of use of the software. The user is responsible for ensuring that the electronic documentation, which is not supplied with the product, matches the product actually used by the user.

1.3.2 Qualified personnel

The product system described in this documentation must only be handled by qualified personal according to the given scope of work. All documentation relevant for the scope of work must be observed, especially the safety and warning notes. Due to its education and experience, qualified personal is able to identify risks and possible dangers when using this products / systems.

1.3.3 Disclaimer

The content of this documentation has been carefully reviewed to comply with the documented hard- and software. We can, however, not exclude discrepancies and do therefore not accept any liability for the exact compliance. This documentation is reviewed regularly. Corrections may be contained in newer versions.

1.4 Preface

1.4.1 Puspose

This reference manual describes the functions and opcodes of the MscDll for the use with the Irinos-System. The target audience is

- software developers, who want to integrate the MscDll into their application software (measurement software) and
- startup technicians and maintenance staff, who need to parametrize the Irinos-System.

1.4.2 Scope of this users manual

This users manual is valid for the industrial measurement system Irinos (IR-xxx) together with the MscDll.

1.4.3 Intended use

Irinos is a flexible High-Speed measurement system for the industrial production measurement technology.

The measurement device is not appropriate for use in medical fields or in explosive areas, for aerospace and for home- or office use. Other fields of application, which are not mentioned but similar, are also excluded from use.

In safety critical areas, the safety in operation must be ensured by external equipment (e.g. external emergency stop).

Please note:

	Warning
	Products from Messtechnik Sachs GmbH must only be used for applications, which are mentioned in the dataheet or in the related documentation. If third party products are used, these must be recommended or permitted by Messtechnik Sachs GmbH. Proper and safe operation of the products require appropriate transportation, storage, mounting, usage and maintenance. Environmental conditions stated in the specification must be observed as well as notes in the related documentation.

1.4.4 Required knowledge

Profound knowledge in PC based software development using Windows is required for integrating and using the MscDll.

1.4.5 Further documentation

Please note the short booklet, which is delivered with each Irinos module. This applies especially to the safety warnings, which are mentioned in it. The specifications of the Irinos-Boxes can be found in the respective datasheet.

Before using the Irinos-System, please read the users manual carefully.

1.4.6 Firmware version

This users manual is related to Firmware version V1.1.

The MscDll

2 The MscDll

The MscDll is the link between the application software (measurement software) and the Irinos-System. It supports the application in the following fields:

- Read measurement values from the Irinos-System.
- Read status information from the Irinos-System.
- Read / write digital in-/output data.
- Parametrize the Irinos-System.

The MscDll uses the Windows API functions for IP based communication, thread management and timing.
Inside the MscDll a separate thread is cunning, which controls the communication. The DLL functions provide data to this thread and vice versa.

Communication to the Irinos-System is based on UDP/IP. The DLL automatically retransmits a data packet, if it has been lost. A direct ethernet connection between the Irinos-System and the PC is adviced. Complex network structures, e.g. routing, tunneling, VPN, etc. are not supported due to timing efficiency.

2.1 Msc.cfg

The MscDll is configured via the configuration file Msc.cfg. This is a text file, which can be opened using a standard editor. The typical content of the Msc.cfg is:

```
[System]
FTDI=OFF
XPort=ON

[XPort]
Address1=192.168.3.99:10002
EnumRetry=2
EnumTimeout=400
SendBufSize=1500
RcvBufSize=65536
```

The parameters are:

Parameter	Description
FTDI	Only for compatibility with older systems. This parameter must have the value OFF.
XPORT	This parameter must have the value ON (enables the IP based communication).
Adress1	IP address of the Irinos-System followed by the port number. The port number is always 10002. Using the port number 10002 requires a UDP send buffer size of 1500 bytes and a receive buffer size of 65536 bytes (see parameters SndBufSize and RcvBufSize). [For compatibility with other systems, the port number 10001 is also supported. In this case the UDP packet size is limited to 800 bytes.]
EnumRetry	Number of retries for searching the Irinos-System at connection startup.
EnumTimeout	Timeout for searching the Irinos-System at connection startup.
SendBufSize	Size of the UDP send buffer. Should always be 1500 bytes.
RcvBufSize	Size of the UDP receive buffer. Should always be 65536 bytes.

If the IP settings of the Irinos-System remain unchanged after

delivery, the Msc.cfg file above can be used directly. Otherwise the IP address must be adopted to the actual IP address of the Irinos-System (Parameter "Address1").

The configuration file Msc.cfg can be created automatically by the Irinos-Tool. Further instructions can be found in the documentation of the Irinos-Tool.

API (programming interface)

3 API (programming interface)

This section describes the function calls of the MscDll.

3.1 Return values (MSC_STATUS)

The function calls of the MscDll almost all have the same return value of the type MSC_STATUS. It is defined as follows:

Return value	Hex representation	Description
MSC_STATUS_SUCCESS	0x00000000	no error detected
MSC_STATUS_FAILED	0xF0000001	general error
MSC_STATUS_INVALID_HANDLE	0xF0000002	call an invalid handle
MSC_STATUS_INVALID_PARAMS	0xF0000003	invalid parameter
MSC_STATUS_NO_RESOURCES	0xF0000004	no resources for creation
MSC_STATUS_NO_DEVICES	0xF0000005	no device found
MSC_STATUS_NOT_INITIALIZED	0xF0000006	member not initialzed
MSC_STATUS_ALREADY_INITIALIZED	0xF0000007	interface is already initialized, repeated call
MSC_STATUS_INVALID_OBJECT_TYPE	0xF0000008	handle identifies an invalid object type, handle is invalid for this function call

Return value	Hex representation	Description
MSC_STATUS_INVALID_CHANNEL_TYPE	0xF0000009	channel does not have the correct type for this function
MSC_STATUS_FUNCTION_NOT_ALLOWED	0xF0000100	function not allowed at this time
MSC_STATUS_NO_DATA_AVAILABLE	0xF0000200	no data available
MSC_STATUS_NO_MORE_DATA	0xF0000400	no more data
MSC_STATUS_BUFFER_TO_SHORT	0xF0000401	buffer too short

3.2 General

3.2.1 MSC_GetVersion

This function returns the current version of the MscDll.

Definition

```
void
MSC_GetVersion(unsigned long* ApiVersion, unsigned long*
DllVersion);
```

Parameter

ApiVersion

Returns the API version of the DLL. The upper 16 bits contain the major version, the lower 16 bits contain the minor version.

DllVersion

Returns the DLL version. The upper 16 bits contain the major version, the lower 16 bits contain the minor version.

Comments

If changes are made to the programming interface that are compatible with previous versions, then the minor version number (low order word) will be incremented. If changes are made that cause an incompatibility with previous versions of the interface then the major version number (high order word) will be incremented.

Applications should check the return value of this function against the MSCAPI_VERSION constant to make sure the DLL supports the expected API version. The major version number must match the expected number exactly. The minor version number needs to be greater than or equal to the expected number.

This function can be called without an active communication.

3.2.2 MSC_WriteCommand

This function writes data to the Irinos-System and waits for a response.

Definition

```
MSC_STATUS
MSC_WriteCommand(
     MSC_HANDLE Handle,
     unsigned char OpCode,
     unsigned long SndBufferSize,
     void* SndBuffer,
     unsigned long RcvBufferSize,
     void* RcvBuffer,
     unsigned long* BytesReceived,
     unsigned long Timeout
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24] .

OpCode

Opcode[52] for the data to be transmitted.

SndBufferSize

Size of the data, which shall be sent.

SndBuffer

Buffer with data, which shall be sent. Minimum size is „SndBufferSize".

RcvBufferSize

Size of the receive buffer. Enough space must be provided for the response data.

RcvBuffer

Buffer for the response data. Minimum size is „RcvBufferSize".

BytesReceived

Number of bytes received.

Timeout

Timeout for the data exchange in ms, e.g. 500.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

This function is used to send a single data packet to the Irinos-System and to receive the related response. The data transfer must be started. The MscDll waits until the data is sent and the response has been received or until a timeout occurs.

See also

Used with the following opcodes: opcRIV[56], opcRMI[58], opcWCC[64], opcRCA[66], opcWCA[69], opcWCL[71], opcRCL[73], opcACL[75], opcDT[76], opcAT[81], opcIT[82], opcSP[83], opcREv[89], opcSAbsT[90], opcWEvCfg[92], opcRSW[101], opcRST[107]

Can be used instead of MSC_SetupStaticChannel[39] & MSC_ReadStatic[41] for the following opcodes: opcRS[96], opcRHS[86], opcBIO[97]

3.3 Connection

3.3.1 MSC_EnumerateDevices

This function counts all configured devices (i.e. Irinos-Systems) and returns the number of systems available.

Definition

```
MSC_STATUS
MSC_EnumerateDevices(
     unsigned long* DeviceNumber
);
```

Parameter

DeviceNumber

Returns the number of systems available.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

The MscDll is able to communicate with multiple Irinos-Systems simultaneously. However, typically only 1 Irinos-System is used, which results in the value 1 for the parameter DeviceNumber.

The behaviour of this function depends on the setup in the configuration file. If new devices are added this function must be called again to open it.

This function does not search for Irinos-Boxes. It searches for Irinos-Systems (Irinos-Boxes with Ethernet interface).

See also

MSC_GetDeviceInfo[23]

MSC_OpenDevice[24]

3.3.2 MSC_GetDeviceInfo

This function returns some information about a device (i.e. about the Irinos-System).

Definition

```
MSC_STATUS
MSC_GetDeviceInfo(
     unsigned long Index,
     unsigned long* BusType,
     char UniqueID[MSC_MAX_UNIQUEID_SIZE]
);
```

Parameter

Index

Device number. The first device has always the number 0. The number of available devices (=Irinos Systems) can be retrieved using MSC_EnumerateDevices[22] .

BusType

Returns the bus type between the PC and the device. The only supported bus type is MSC_TYPE_SOCKET_XPORT (value 0x00000001).

UniqueID

Returns a 0-terminated ASCII string containing the IP network address.
MSC_MAX_UNIQUEID_SIZE has the value 40, thus the maximum string length is 40 bytes.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

In order to provide backward compatibility to older systems, the DLL also supports the bus type MSC_TYPE_USB_FTDI (value 0x00000002). This bus type is no more relevant for newer systems.

See also

MSC_EnumerateDevices[22]

3.3.3 MSC_OpenDevice

This function opens the connection to a device (i.e. to an Irinos-System).

Definition

```
MSC_STATUS
MSC_OpenDevice(
     unsigned long Index,
     MSC_HANDLE* Handle
);
```

Parameter

Index

Device number. The first device has always the number 0.
The number of available devices (=Irinos Systems) can be retrieved using MSC_EnumerateDevices[22].

Handle

Returns a device-handle.
This handle is required by various DLL functions in order to communicate with the device.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Even though is makes no sense, multiple handles to the same device can be created.

See also

MSC_EnumerateDevices[22]

3.3.4 MSC_InitDevice

This function initializes the internal device data.

Definition

```
MSC_STATUS
MSC_InitDevice(
     MSC_HANDLE Handle
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

All channel information, notifications and error counters will be reset.

See also

MSC_OpenDevice[24]

3.3.5 MSC_CloseDevice

This function closes a device-handle.

Definition

```
MSC_STATUS
MSC_CloseDevice(
     MSC_HANDLE Handle
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

If the last handle to a device is closed, all internal information about the device state will be cleared.

See also

MSC_OpenDevice[24]

3.3.6 MSC_Start

This function starts the data transfer.

Definition

```
MSC_STATUS
MSC_Start(
     MSC_HANDLE Handle,
     unsigned long SndPeriod,
     unsigned long DisconnectTimeout,
     unsigned long RetryCount,
     unsigned long ResponseTimeout
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

SndPeriod

[ms] Time-period between two message frames, which will be sent to the device. If the period is smaller than the time is required to get the answer from the device, the next message frame is sent as soon as possible.

DisconnectTimeout

[ms] If no answer from the device is received in this time interval the device notification[32] is set.

RetryCount

Number of retries before a communication error is reported.

ResponseTimeout

[ms] Time interval before a retry is started.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

Communication will only work, if the data transfer has been started before using this function.

Example values

The following values are appropriate for static measurement and for most dynamic measurements:

SndPeriod = 1
DisconnectTimeout = 500
RetryCount = 10
ResponseTimeout = 75

In order to achieve a short send period, a 1ms Multimedia-Timer must be started in the Windows application. It decreases the Windows Tick-Time from the standard value (15,6ms) to ≤ 1ms. It is suggested to implement the Multimedia-Timer into the application.

All example values are appropriate for a direct Ethernet connection between the PC and the Irinos-System.

See also

MSC_OpenDevice 24

MSC_Stop 29

3.3.7 MSC_Stop

This function stops the data transfer.

Definition

```
MSC_STATUS
MSC_Stop(
     MSC_HANDLE Handle
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24] .

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Siehe auch

MSC_OpenDevice[24]

MSC_Start[27]

3.3.8 MSC_GetDeviceState

This function returns the communication status to the Irinos-System. It serves only as a diagnostic tool. For normal operation, it is not required. Its implementation is suggested in order to have extended diagnostic capabilities in case of communication problems.

Definition

```
MSC_STATUS
MSC_GetDeviceState(
     MSC_HANDLE Handle,
     unsigned long* LastMsgReceived,
     unsigned long* SndErrorCounter,
     unsigned long* RcvErrorCounter,
     unsigned long* CmdDiscarded,
     unsigned long* CmdDiscardedArray,
     unsigned long Flags
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice.

LastMsgReceived

Returns the time since the reception of the last data frame.

SndErrorCounter

Returns the error counter of data frames, which could not be sent. The counter is set to 0 either by the function MSC_Start 27 or if the flags value MSC_RESET_ERROR_COUNTERS is set.

RcvErrorCounter

Returns the error counter of data frames, which could not be received completely. The counter is set to 0 either by the function MSC_Start 27 or if the flags value MSC_RESET_ERROR_COUNTERS is set.

CmdDiscarded

Returns a counter which is increased each time if an opcode is received which was not expected. This counter is the summary of the CmdDiscardedArray. The counters will be set to zero by the function MSC_Start 27 or if the flags value MSC_RESET_DISCARDED_COUNTERS is set.

CmdDiscardedArray

Pointer to an user provided array of 256 unsigned long values. Each value contains the error counter for an opcode specified by index. The counters will be set to zero while function MSC_Start 27 or if the flags value MSC_RESET_DISCARDED_COUNTERS is set.

Flags

Contains bitwise information for the MscDll. The flags may be combined.

MSC_RESET_ERROR_COUNTERS
This bit clears the send- and receive error counter.

MSC_RESET_DISCARDED_COUNTERS
This bit clears the counter for invalid telegrams.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code will be returned.

See also

MSC_OpenDevice[24]

MSC_Start[27]

3.4 Notifications

Using notifications, the application can be informed about the following events:

a) Connection failure („DisconnectTimeout", see MSC_Start[27])

b) New data available for a static data channel (see MSC_SetupStaticChannel[39])

c) Receive buffer for dynamic measurement full.

The use of notifications is recommended but not required.

Using notifications, it is ***very important that the notification itself and reading, processing and displaying data is handled in a separate threads***. Otherwise this may lead to communication delays or even to a breakdown of the communication.

The following approach is suggested:

- In case a notification occurs, a flag is set.
- This flag is checked cyclically in a thread. If it is set, new data is read and processed and the flag is reset.

This cyclic check can for example be implemented into a 30ms timer-event of the GUI.

It is recommended to use messages for notifications (MSC_SetNotificationMessage[33]).

3.4.1 MSC_SetNotificationMessage

This function enables sending Windows messages as notifications.

Details for receiving Windows messages can be found in the documentation of your development tools.

Definition

```
MSC_STATUS
MSC_SetNotificationMessage(
     MSC_HANDLE Handle,
     int OpCode,
     HWND hWnd,
     ULONG MsgCode,
     ULONG wParam,
     ULONG lParam
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24] .

Opcode

- Opcode of the static channel (opcRS[96], opcRHS[86] oder opcBIO[97]). A notification message will be sent, if new data has arrived for the respective opcode.
- -1 for a notification in case of a communication timeout (see MSC_Start[27]).
- The opcode of the dynamic measurement (opcRDM1[106] oder opcRDM2[106]). If the receive buffer of the respective dynamic measurement is full, a notification message will be sent.

hWnd

A Windows-handle, which receives the Windows message.

MsgCode

Message number, which must be defined by the application (see also comment below).

wParam

The wParam of the Windows message.

lParam

The lParam of the Windows message.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

The notification is only used for devices and for static and dynamic data transfer channels.

A device notification reports about a communication error.

For a static data transfer channel, the notification is sent each time new data has arrived (-> once per send period).

For a dynamic data transfer channel, the notification is only sent, if the receive buffer is completely full.

The notification can be cleared, if the function is called with the parameter hWnd = NULL.

MsgCode

The message-number is defined by the application. Using Visual C++, this can be done for example as follows:

```
#define WM_MESSAGE_MSC_READSTATIC    (WM_USER + 0)
#define WM_MESSAGE_MSC_BITIO         (WM_USER + 1)
#define WM_MESSAGE_MSC_HW_STATUS     (WM_USER + 2)
```

See also

MSC_OpenDevice[24]

MSC_Start[27]

3.4.2 MSC_SetNotificationEvent

This function registers a event for notification.

Details for using Events can be found in the documentation of your development tools.

Definition

```
MSC_STATUS
MSC_SetNotificationEvent(
     MSC_HANDLE Handle,
     int OpCode,
     HANDLE Event
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

- Opcode of the static channel (opcRS[96], opcRHS[86] oder opcBIO[97]). A notification message will be sent, if new data has arrived for the respective opcode.

- -1 for a notification in case of a communication timeout (see MSC_Start[27]).
- The opcode of the dynamic measurement (opcRDM1[106] oder opcRDM2[106]). If the receive buffer of the respective dynamic measurement is full, a notification message will be sent.

Event

Initialized event handle.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

The notification is only used for devices and for static and dynamic data transfer channels.

A device notification reports about a communication error.

For a static data transfer channel, the notification is sent each time new data has arrived (-> once per send period).

For a dynamic data transfer channel, the notification is only sent, if the receive buffer is completely full.

The notification can be unregistered if the function is called with Event == NULL.

The MscDll never resets this event.

See also

MSC_OpenDevice[24]

MSC_Start[27]

3.4.3 MSC_SetNotificationCallback

This function registers a callback function for notification.

Definition

```
MSC_STATUS
MSC_SetNotificationCallback(
     MSC_HANDLE Handle,
     int OpCode,
     MSC_NOTIFICATION_CALLBACK* CallbackFunction,
     void* NotificationContext
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

- Opcode of the static channel (opcRS[96], opcRHS[86] oder opcBIO[97]). A notification message will be sent, if new data has arrived for the respective opcode.
- -1 for a notification in case of a communication timeout (see MSC_Start[27]).
- The opcode of the dynamic measurement (opcRDM1[106] oder opcRDM2[106]). If the receive buffer of the respective dynamic measurement is full, a notification message will be sent.

CallbackFunction

The address of the callback function. Use NULL to unregister the callback function.

NotificationContext

A caller provided context pointer which is passed unchanged to the callback function[38].

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code will be returned.

Comments

The notification is only used for devices and for static and dynamic data transfer channels.

A device notification reports about a communication error.

For a static data transfer channel, the notification is sent each time new data has arrived (-> once per send period).

For a dynamic data transfer channel, the notification is only sent, if the receive buffer is completely full.

The notification can be unregistered if the function is called with Event == NULL.

The callback function is called from the communication thread. Only very limited code can be executed in the callback-function. Otherwise this may lead to communication delays or even to a breakdown of the communication.

See also

MSC_OpenDevice 24

MSC_Start 27

Callback function

This function is the prototype for a callback notification.

Definition

```
void
MSC_NOTIFICATION_CALLBACK(
     void* NotificationContext
);
```

Parameter

NotificationContext

This parameter is the same which was passed to the function MSC_SetNotificationCallback[37]. The application can store a context information in this pointer.

Comments

This function is called in a different thread context. The application must handle the synchronisation.

3.5 Static transfer channels

3.5.1 MSC_SetupStaticChannel

This function initializes a static data transfer channel. It is used to exchange data between the MscDll and the Irinos-System automatically and continuously.

Definition

```
MSC_STATUS
MSC_SetupStaticChannel(
     MSC_HANDLE Handle,
     unsigned char OpCode,
     unsigned long SndBufferSize,
     void* SndBuffer,
     unsigned long RcvBufferSize
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

Opcode for the static channel:
opcRS[96] for reading static measurement values.
opcRHS[86] for reading the hardware status.
opcBIO[97] for exchanging Bit I/O data.

SndBufferSize

Size of the data, which shall be sent.

SndBuffer

Buffer with data, which shall be sent. Minimum size is „SndBufferSize".

RcvBufferSize

Maximum size of the receive data.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

The MscDll uses an internal buffer for the receive data. It is overwritten if new data is received. The application has no access to this internal buffer. It needs to be read using the function MSC_ReadStatic[41].

The opcode is either opcRS[96], opcRHS[86] or opcBIO[97].

The send buffer is not used for opcRS[96]. Nevertheless, it must always be defined having a minimum size of 1 byte.

If data in the send buffer is changes, the MscDll must be informed via the function MSC_RefreshChannel[43].

See also

MSC_OpenDevice[24]

MSC_Start[27]

MSC_ReadStatic[41]

MSC_RefreshChannel[43]

3.5.2 MSC_ReadStatic

This function reads new data from a static data transfer channel.

Definition

```
MSC_STATUS
MSC_ReadStatic(
     MSC_HANDLE Handle,
     unsigned char OpCode,
     unsigned long BufferSize,
     void* Buffer,
     unsigned long* DataCount
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

Opcode for the static channel:
opcRS[96] for reading static measurement values.
opcRHS[86] for reading the hardware status.
opcBIO[97] for exchanging Bit I/O data.

BufferSize

Size of the destination buffer ("Buffer") in bytes.

Buffer

Destination buffer. Data will be copied into this buffer. Its size must be at least "BufferSize".

DataCount

Number of bytes, which have been received recently. 0 if no new data has been received.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

A static measurement channel needs to be initialized with the function MSC_SetupStaticChannel[39]. The opcode for Msc_ReadStatic is the same as it has been used for MSC_SetupStaticChannel[39].

See also

MSC_OpenDevice[24]

MSC_SetupStaticChannel[39]

3.5.3 MSC_RefreshChannel

This function updates the send buffer of the given opcode.

Definition

```
MSC_STATUS
MSC_RefreshChannel(
     MSC_HANDLE Handle,
     unsigned char OpCode
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

The opcode, whose send buffer needs to be updated (opcBIO[97]).

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

This function is used to update the output data inside the MscDll. Currently the only application is updating digital outputs (opcBIO[97]).

[The output data of the opcode opcRHS[86] does not need to be updated, since it remains unchanged.]

This function does not start a data transfer on the network interface. It only informs the MscDll about new data.

See also

MSC_OpenDevice[24]

MSC_SetupStaticChannel[39]

3.6 Transfer channels for dynamic measurement values

3.6.1 MSC_SetupExtendedDynamicChannel

This function initializes a data transfer channel for dynamic measurement values.

Definition

```
MSC_STATUS
MSC_SetupExtendedDynamicChannel(
     MSC_HANDLE Handle,
     unsigned char OpCode,
     unsigned char NumberOfSubChannels,
     unsigned long SndBufferSize,
     void* SndBuffer
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

The opcode of the dynamic measurement (opcRDM1[106] or opcRDM2[106]).

NumberOfSubChannels

Number of measurement channels, which are transferred in a data frame (= number of measurement channels used for the dynamic measurement).

SndBufferSize

The size of the send buffer must be 1.

SndBuffer

Pointer to the send buffer. Minimum length is 1 byte. Its content is not relevant. It is only required to keep to the data structure.

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

An extended dynamic measurement channel stores the data in several FIFO like buffers. The buffers are provided by the application with the function MSC_AttachSubChannelBuffer[46].

The measurement is running if a buffer is attached to all of the subchannels and these buffers are not full.

All measurement values have the data type 32 bit signed long. This also applies to measurement channels, which have a 16 or 8 bit value range.

See also

MSC_OpenDevice[24]

MSC_AttachSubChannelBuffer[46]

MSC_GetPosition[49]

3.6.2 MSC_AttachSubChannelBuffer

This function adds buffers for measurement values to a dynamic data transfer channel. One buffer must be provided per measurement channel, which is used in the dynamic measurement.

Definition

```
MSC_STATUS
MSC_AttachSubChannelBuffer(
     MSC_HANDLE Handle,
     unsigned char OpCode,
     unsigned char SubChannel,
     unsigned long BufferSize,
     void* Buffer
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

The opcode of the dynamic measurement (opcRDM1[106] or opcRDM2[106]).

SubChannel

Number of the measurement channel. The number of the first measurement channel is 0. It must be smaller than "NumberOfSubChannels" in the function MSC_SetupExtendedDynamicChannel[44].

BufferSize

Size of the measurement buffer. Each measurement value requires 4 bytes.
If for example 1.000 measurement values per channel shall be sampled, a buffer size of 4.000 bytes is required.

Buffer

Pointer to the buffer for the measurement values. Its size must be at least "BufferSize".

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

The memory for the buffer must be allocated by the application. It must be permanent, until it is detached using the function MSC_DetachSubChannelBuffers[47].

This function must be called once per measurement channel used for the dynamic measurement.

The opcode is the same, as it has been used for MSC_SetupExtendedDynamicChannel[44].

See also

MSC_OpenDevice[24]

MSC_SetupExtendedDynamicChannel[44]

MSC_DetachSubChannelBuffers[47]

MSC_GetPosition[49]

3.6.3 MSC_DetachSubChannelBuffers

This function detaches all buffers for measurement values from a dynamic data transfer channel.

Definition

```
MSC_STATUS
MSC_DetachSubChannelBuffers(
     MSC_HANDLE Handle,
     unsigned char OpCode
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

The opcode of the dynamic measurement (opcRDM1[106] or opcRDM2[106]).

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

The function detaches all buffers attached to the subchannels of an extended dynamic channel by calling MSC_AttachSubChannelBuffer[46]. If the buffers are detached, the extended dynamic measurement inside the MscDll stops.

See also

MSC_OpenDevice[24]

MSC_SetupExtendedDynamicChannel[44]

MSC_AttachSubChannelBuffer[46]

MSC_GetPosition[49]

3.6.4 MSC_GetPosition

This function returns the position of a dynamic measurement. This allows determining the number of measurement values, which have already been sampled and transferred to the MscDll.

Definition

```
MSC_STATUS
MSC_GetPosition(
     MSC_HANDLE Handle,
     unsigned char OpCode,
     unsigned long* Position
);
```

Parameter

Handle

Device-Handle, which has been returned by a previous call of the function MSC_OpenDevice[24].

Opcode

The opcode of the dynamic measurement (opcRDM1[106] or opcRDM2[106]).

Position

Returns the position of the dynamic measurement in bytes (see also comment below).

Return value

If successful, MSC_STATUS_SUCCESS will be returned. In case of an error, an error code[18] will be returned.

Comments

Measurement values are always stored as 32 bit values. In order to get the number of measurement values, the variable "Position" must be divided by 4.

Example: If a buffer is filled with 400 bytes, it contains 100 measuring values.

See also

MSC_OpenDevice 24

MSC_SetupExtendedDynamicChannel 44

MSC_AttachSubChannelBuffer 46

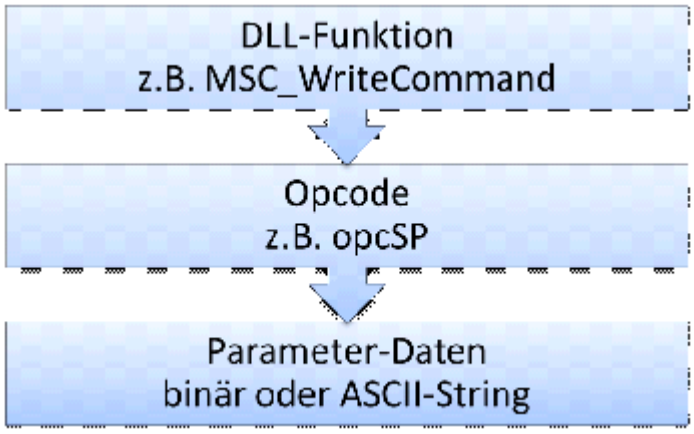
Opcodes and parameters

4 Opcodes and parameters

4.1 The role of opcodes

The application software communicates with the Irinos system via the MscDll. The MscDll functions are called from the application software. Most of the MscDll's functions need additional data. Some functions (like MSC_WriteCommand[20]) are just carriers for data.

The meaning of the additional data is defined in opcodes which in turn use parameter data to completely define their function. Parameter data can be binary or an ASCII-type string. Each opcode is assigned to a specific function.



4.2 Opcode overview

Opcodes "Initialization"

Opcode	Hex value	Function	Parameter type
opcRIV[56]	0x01	Read inventory (number of boxes)	String[56]
opcRMI[58]	0x03	Read Box information (digital type	String[56]

Opcode	Hex value	Function	Parameter type
		plate)	
opcRSS 63	0x05	Read System-String	String 56
opcWCC 64	0x09	Write channel characteristics	String 56
opcRCA 66	0x10	Read channel assignment	String 56
opcWCA 69	0x11	Write channel assignment	String 56

Opcodes "Configuration and miscellaneous"

Opcode	Hex value	Function	Parameter type
opcWCL 71	0x22	Write channel list	String 56
opcRCL 73	0x23	Read channel list	String 56
opcACL 75	0x24 (0x26)	Activate channel list for static measurement	String 56
opcDT 76	0x30	Define trigger for dynamic measurement	String 56

Opcode	Hex value	Function	Parameter type
opcAT 81	0x31	Activate trigger for dynamic measurement	String 56
opcIT 82	0x32	Inactivate trigger for dynamic measurement	String 56
opcSP 83	0x35	Set (channel-) parameter	String 56
opcRHS 86	0x38	Read hardware status of measurement channels	Binary
opcREv 89	0x39	Read current event of the Irinos-Boxes	Binary
opcSAbsT 90	0x3A	Set absolute time (for diagnostics memory)	String 56
opcWEvCfg 92	0x3D	Write event configuration	String 56
opcClrEv 94	0x3E	Clear event	String 56

Opcodes "Measurement"

Opcode	Hex value	Function	Parameter type
opcRS 96	0x40	Read static measurement values	Binary
opcBIO 97	0x42	Read digital inputs / write digital outputs	Binary
opcBIORO 99 (Available from Firmware version 1.4.x.x)	0x43	Read current state of digital in-/outputs	Binary
opsRSW 101	0x44	Read statusword for dynamic measurement	Binary
opcDDM1 104	0x50	Define dynamic measurement 1	String 56
opcDDM2 104	0x51	Define dynamic measurement 2	String 56
opcRDM1 106	0x60	Read values of dynamic measurement 1	Binary
opcRDM2 106	0x61	Read values of dynamic measurement 2	Binary

Opcodes "Service"

Opcode	Hex value	Function	Parameter type
opcRST 107	0x7E	System Reset	String 56

4.3 Parameter type "String"

For opcodes with the parameter type "string", the following rules apply:

a) Only ASCII characters in the range 0x20 .. 0x7F must be used. The extended ANSII characters or "wide strings (16 bits)" are not supported.

b) Everey string is framed by one leading and one trailing ‚#' character. This characters are stripped before scanning the string and does not carry any information.

c) Immediately after the leading ‚#' character follows the command code. This is a variable length substring that must match the predefined command exactly. Upper and lower case are important.

d) Items within a string are separated by semicolons.

e) Unused parts of a parameter string must contain the ‚*' character. The semicolon must always be present.

4.4 Opcodes: Initialization

4.4.1 opcRIV: Read inventory (number of boxes)

This opcode allows reading the number of available Irinos-Boxes from the Irinos-System.

Overview

Opcode: 0x01

Name: opcRIV

Function: Read inventory (number of boxes)

Parameter type: String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

None. No data needs to be sent.

Response string from the Irinos-System

#{Number of Irinos-Boxes};{Number of modules}#

Number of Irinos-Boxes

Total number of Irinos-Boxes used in the Irinos-System.

Number of modules

This value is identical with "Number of Irinos-Boxes". It is only used for backward compatibility to older systems.

Example:

#3;3# -> 3 Irinos-Boxes are used.

This opcode cannot return an error response string.

Comment

The Master-Box (with Ethernet interface to the PC) is also counted. The example above could be a combination of the following Irinos-Boxes:

1 Master-Box IR-MASTER-KB1-68-68-SYSP-ETHIL
1 Slave-Box for inductive probes IR-TFV-8-TESA-M16-IL
1 Slave-Box for incremental encoders IR-INC-4-SEL1VSS-D15F-IL

4.4.2 opcRMI: Read Box information (digital type plate)

This opcode allows reading detailed Box information (digital type plate).

Overview

Opcode:	0x03
Name:	opcRMI
Function:	Read module information
Parameter type:	String 56

DLL-Function

MSC_WriteCommand 20

Request string to the Irinos-System

`#{Box number};{Value 2}#`

z.B. `#0;2#`

Box number

Number (address) of the Irinos-Box, starting with 0.

0 is always the Master-Box.

≥ 1 is a Slave-Box

To request the Box information of all Irinos-Boxes, the request must be repeated once for each Irinos-Box.

Value 2

Fixed value "2". If this part of the parameter string is missing, an older version of the response string is returned (backward compatibility).

Response string from the Irinos-System

```
#{Box number};{Device string};{MAC address};{Serial};
{Production code};{Hardware version};{Hardware revision};
{Firmware version};{Sample period};{Total number of measurement
channels};{Number of 64bit channels};{Number of 32bit
channels};{Number of 16bit channels};{Number of 8bit channels};
{Always 0};{Always 0};{Always 0};{Always 0};{Always 0};{Number
of digital inputs};{Number of digital outputs};{GUID};{Box
name};{Order number}#
```

Example:

```
#0;0;IR-TFV-8-IET-M16-ETHIL;A0-BB-3E-E0-00-03;I123456;S-W3-
28;HW V1.1;HWRev 1;SW V1.0.0.27;50;8;0;0;8;0;0;0;0;0;0;2;0;
{0C003B23-2C74-49A0-BCB1-E81C7C32C42A};LBox 0;828-5006#
```

Box number (1)

e.g.: `0`

Number (address) of the Irinos-Box. Same as in request string.

Device string (2)

e.g. `IR-TFV-8-IET-M16-ETHIL`

Type string of the Irinos-Box (like it is printed on the type plate).

MAC address (3)

e.g. `A0-BB-3E-E0-00-03`

MAC address of the Irinos-Box.

Each Irinos-Box has a unique MAC address, which is assigned during production. It is also printed on the type plate.

Master-Boxes as well as Slave-Boxes are equipped with a MAC address. The Master-Boxes also use it for the Ethernet interface.

Serial (4)

e.g. `I123456`

The serial number is assigned to the Irinos-Box during production. It is also printed on the type plate.

Production code (5)

e.g. `S-W3-28`

Internal production code of the manufacturer.

Hardware version (6)

e.g. `HW V1.1`

Version of the electronics.

Hardware revision (7)

e.g. `HWRev 1`

Compatibility code between the hardware and the firmware. It ensures that a firmware update is only allowed if the firmware version is compatible to the hardware revision.

Firmware version (8)

e.g. `SW V1.0.0.27`

Firmware version.

The first part of the firmware version is incremented in case of major changes.

The second part of the firmware version is incremented in case new functionality has been implemented.

The third part of the firmware version is incremented in case one or more bugs were fixes.

The fourth part of the firmware version is an internal counter.

Sample period (9)

e.g. `50`

Sample period in µs.

The sample period is constant for each Irinos-Box. It is independent of the sample period of the static or dynamic measurement.

The time period for a dynamic measurement must be a multiple of the sample periods of the involved Irinos-Boxes. See also opcode opcDT 76.

Total number of measurement channels (10)

e.g. `8`

Total number of measurement channels of a Irinos-Box.

Number of 64 bit channels (11)

`always 0`

For future use. Currently 64 bit channels are not supported.

Number of 32 bit channels (12)

e.g. `0`

Number of 32 bit measurement channels an Irinos-Box has. For example channels for incremental encoders.

Number of 16 bit channels (13)

e.g. `8`

Number of 16 bit measurement channels an Irinos-Box has. For example measurement channels for inductive probes or analogue inputs.

16 bit values are always transferred as 32 bit values to the MscDll / application.

Number of 8 bit channels (14)

e.g. `0`

Number of 8 bit measurement channels an Irinos-Box has. Currently no such Irinos-Box is available.

8 bit values are always transferred as 32 bit values to the MscDll / application.

Parameters 15 - 19

Always 0. For future use.

Number of digital inputs (20)

e.g. `2`

Number of digital inputs the Irinos-Box has.

For the communication to the MscDll, the number of digital inputs is always rounded up to a multiple of 8.
In the example 2 digital inputs are available. These are rounded up to 8, whereas the inputs 3-8 are always low.

Number of digital outputs (21)

e.g. `0`

Number of digital outputs the Irinos-Box has.

Like the digital inputs, they are always rounded up to a multiple of 8.
If an Irinos-Box for example has 4 digital outputs, the "virtual" outputs 5-8 will be added. They can be addressed by the MscDll, but this has no effect.

GUID (22)

e.g. `0C003B23-2C74-49A0-BCB1-E81C7C32C42A`

For future use.

Box name (23)

e.g. `LBox 0`

For future use.

Order number (24)

e.g. `828-5006`

Order number of the Irinos-Box.

Response string from the Irinos-System in case of an error

`#-99#` General syntax error of the request string.

`#-1#` if the box number in the request string is invalid.

4.4.3 opcRSS: Read System-String

Via this opcode a string can be requested, which represents the structure of the Irinos-System.

Overview

Opcode:	0x05
Name:	opcRSS
Function:	Read system string
Parameter type:	String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

```
#1#
```

Response string from the Irinos-System

```
#{Value 1};{Number of Irinos-Boxes};{Order-No. Irinos-Box 0};
{Order-No. Irinos-Box 2};...;{Order-No. Irinos-Box n}#
```

Example for an Irinos-System with the Irinos-Boxes IR-INC (Box 0, Order-No. 828-5013) and IR-TFV (Box 1, Order-No. 828-5003):
`#1;2;828-5013;828-5003#`

Value 1

Always 1.

Number of Irinos-Boxes

e.g. `2`

Number of Irinos-Boxes used in the Irinos-System.

Order-No. Irinos-Box n

e.g. `828-5013`

The order numbers of all Irinos-Boxes are listed in the order of their addresses.

Response string from the Irinos-System in case of an error

`#-99#` General syntax error of the request string.

`#-1#` if the request string does not contain the value 1.

Comments

It is recommended to store the expected response string in the measurement software (e.g. in its Ini file). After establishing the connection, the string requested from the Irinos-System is compared to the reference string. This allows verifying if all required Irinos-Boxes are available and properly connected.

4.4.4 opcWCC: Write channel characteristics

This opcode is used for parametrizing a measurement channel. The parameters depends on the channel type. Currently this opcode is only used for input channels for incremental encoders of the Irinos-Box IR-INC.

Overview

`Opcode:` `0x09`

Name: opcWCC

Function: Write channel characteristics

Parameter type: String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

#{ChannelName};{Configuration type};{Store}#

e.g. #T10;TTL;1#

ChannelName

Name of the measurement channel.

After startup all measurement channels are enumerated in ascending order with the names T1 T2 T3 T4 ... Tn.

The name of the measurement channel can be changed by the application via the opcode opcWCA[69]. In this case the newly assigned channel name must be used.

Configuration type

„1VSS" to change an incremental channel type to 1Vpp.

„TTL" to change an incremental channel type to TTL / RS422.

Store

„0": The change remains active until the next system restart. Afterwards the old configuration becomes active again.

„1": The change applies permanently.

Please note: The number of write operations into the nonvolatile memory is limited. Further information can be found in the users manual.
If a measurement channel is reconfigured for the same type as it already was, no store operation is executed -> it is for example possible to configure the same channel for the same type multiple times after startup by the application.

Response string from the Irinos-System in case an incremental channel was addressed (IR-INC)

`#0#` Success

`#-n#` Parameter ‚n' of the request string invalid

`#-99#` General syntax error of the request string

Response string from the Irinos-System in case any other measurement channel was addressed

---

`#-98#`

Comments

A measurement channel for incremental encoders can also be configured for 1Vpp or TTL/RS422 via the Irinos-Tool.

After configuration the position of the incremental input channel is reset.

By reconfiguring an incremental input channel after start of the application software, it does not matter how an Irinos-Box is pre-configured. This allows a quick replacement of an Irinos-Box without manual reconfiguration of the "new" Irinos-Box.

4.4.5 opcRCA: Read channel assignment

This opcode allows reading the current assignment of the measurement channels to the physical measurement inputs. ***Most applications do not need this opcode.***

Overview

Opcode: 0x10

Name: opcRCA

Function: Read channel assignment

Parameter type: String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

#{Segmentindex}#

e.g. #1#

Segmentindex

See description of the segments.

Response string from the Irinos-System

List See example

#-1# Invalid Segment-Index

#-99# General syntax error of the request string

Example response (list)

```
#1;1;
T1,1,0,1,1;
T2,2,0,1,2;
T3,3,0,1,3;
T4,4,0,1,4;
T5,5,1,1,1;
T6,6,1,1,2;
T7,7,1,1,3;
T8,8,1,1,4;
T9,9,1,1,5;
T10,10,1,1,6;
T11,11,1,1,7;
T12,12,1,1,8
#
```

The first block is `#{Segment};{NumberOfSegments}#`

e.g. `1;2`

Each additional block defines a logic measurement channel. The meanings of the values are:

T1,1 [a] Channel name (ASCII string, max. 4 characters)
,0,1,
1

T1, [b] Logic channel number in ascending order
1,0,
1,1

T1,1 [c] Number of the Irinos-Box (Box address)
,
0,1,
1

T1,1 [d] Modul-ID. Always 1. For backward compatibility with older
,0, systems.
1,1

T1,1 [e] Physical channel number within the Irinos-Box [c]. Starts
,0,1,with 1.
1

Segments

For Irinos-Systems with many measurement channels, the complete list cannot be transferred in a single data frame to the PC. Hence it is split up into segments.

If the first segment is requested using the segment-index #1#, the response starts with the same segment-index followed by the total number of segments (e.g. #1;2;). To get the full channel assignment list, this request must be resent with an increasing segment-index, until the total number of segments has been reached.
In case the list is for example split up into 2 segments, the first part must be requested using the segment index #1# and the second part must be requested using the segment index #2#.

By definition the maximum segment size is 32 measurement channels.

Comments

During startup of the Irinos-System, the channel assignment list is created automatically. This list is dynamic, it adapts automatically to the available Irinos components.

The channel assignment list corresponds to the channel list 0, which can be requested via the opcode opcRCL73. It can even be used as a channel list for a dynamic measurement. However, this is not recommended. Use a separate channel list.

4.4.6 opcWCA: Write channel assignment

This opcode allows changing the current assignment of the measurement channels to the physical measurement inputs. It is mainly implemented for backward compatibility to older systems. ***It is not required for new applications.***

Overview

Opcode:	0x11
Name:	opcWCA
Function:	Write channel assignment
Parameter type:	String56

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

Example:

```
#T1,1,0,1,1;T2,2,0,1,2;T3,3,0,1,3#
```

Assignment (1. part of the example):

T1, [a] Channel name (ASCII string, max. 4 characters)
1,0,
1,1

T1, [b] Logic channel number in ascending order
1,0,
1,1

T1,1[c] Number of the Irinos-Box (Box address)
,
0,1,
1

T1,1[d] Modul-ID. Always 1. For backward compatibility with older
,0, systems.
1,1

T1,1[e] Physical channel number within the Irinos-Box [c]. Starts
,0,1 with 1.
,**1**

Response string from the Irinos-System

`#0#` Success

`#-1#` Channel name (e.g. T1) is larger than 4 characters

`#-2#` Invalid logic channel number

`#-3#` Invalid Irinos-Box number (Box address)

`#-4#` Invalid module ID

`#-5#` Physical channel number invalid

`#-6#` Not enough parameters

`#-7#` No semicolon

`#-99#` General syntax error of the request string

Comments

During startup of the Irinos-System, the channel assignment list is created automatically. This list is dynamic, it adapts automatically to the available Irinos components.

Via this opcode the generated list can be overwritten.

The string length of a single data frame is limited. For Irinos-Systems with many measurement channels (> 32) writing the channel assignment must be split up into multiple steps. It is suggested to write a maximum of 32 logic channels at once.

Example: 42 logic channels shall be written. First write the logical channels 1 - 32 and afterwards the logical channels 33 - 42.

The logical channel numbers need to be written in ascending order.

4.5 Opcodes: Configuration and miscellaneous

4.5.1 opcWCL: Write channel list

This opcode writes a channel list to the Irinos-System.

A valid channel list is always required for dynamic measurement. It can further be used to select channels for static measurement (not required).

Overview

Opcode: 0x22

Name: opcWCL

Function: Write channel list

Parameter type: String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

#{Number of channel list};{Channel name 1};…;{Channel name n}#

Example:

#2;T1;T2;T5;T18#

Number of channel list

Number of the channel list, which shall be written. Permissible values: 1..10

Channel name

Name of the measurement channel according to the channel assignment.

During startup all measurement channels are named in ascending order: T1, T2, T3, ...

The channel names can be changed via the channel assignment (opcWCA[69]).

Response string from the Irinos-System

`#0#` Success

`#-n#` Invalid parameter no `n`

`#-99#` General syntax error of the request string

Comments

Channel lists contain all or a selection of the available measurement channels.

They are used to select the measurement channels, which shall be used for a dynamic measurement (see opcDDMx[104]). If a channel list is changed while a dynamic measurement is active, this has no effect on the dynamic measurement. It will be used if the next dynamic measurement is started.
The maximum number of channels allowed for dynamic measurement is 32.

Channel lists can be used to reduce the number of measurement channels used for static measurement (see opcACL[75]). For systems with less than about 64 measurement channels, this makes no sense. If more measurement channels are available, this can speed up transferring dynamic measurement values from the Irinos-System to the PC.
If the channel list used for static measurement is changed, it becomes active immediately.

The channel list 0 corresponds to the channel assignment (see opcRCA[66]) and contains all measurement channels available. It can only be changed via the opcode opcWCA[69].
All other channel lists (1..10) can be overwritten at any time.

4.5.2 opcRCL: Read channel list

This opcode allows reading a channel list from the Irinos-System.

Overview

`Opcode:`	`0x23`
`Name:`	`opcRCL`
`Function:`	`Read channel list`

Parameter type: String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

#{Number of channel list}#

Example: #3#

Number of channel list

Number of the channel list, which shall be readout. Permissible values: 0..10

Response string from the Irinos-System

Liste See description below

#-n# Invalid parameter no n

#-99# General syntax error of the request string

Description of the channel list

#{Number of channel list};{Channel name 1};…;{Channel name n}#

Beispiel: #2;T1;T2;T5;T18#

Number of channel list

Number of the channel list, which has been readout.

Channel name

Name of the measurement channel.

Comments

After startup all channel lists are automatically created. Each contains all measurement channels.

Prior to a dynamic measurement, the channel list used for the dynamic measurement must be written with the selected channels (see opcWCL[71]).

Channel lists contain all or a selection of the available measurement channels.

The channel list 0 corresponds to the channel assignment (see opcRCA[66]) and contains all measurement channels available.

4.5.3 opcACL: Activate channel list for static measurement

Via this opcode a channel list can be selected for static measurement (only useful is many measurement channels are available).

Overview

```
Opcode:           0x24
                  (0x26 also allowed for backwards
compatibility; not to be used in new applications)

Name:             opcACL

Function:         Activate channel list

Parameter type:   String[56]
```

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

`#{Number of channel list}#`

Example: `#2#`

Number of channel list

Number of the channel list, which shall be used for static measurement. Permissible values: 0..10

Response string from the Irinos-System

`#0#` Success

`#-1#` Invalid number of channel list

`#-99#` General syntax error of the request string

Comments

This opcode is only required for applications with many measurement channels, if a dynamic measurement with high a sample rate and many channels is used in parallel.

Typically the pre-defined channel list 0 is used for static measurement. It contains all measurement channels.

This opcode has no effect on the channel list used for dynamic measurement.

4.5.4 opcDT: Define trigger for dynamic measurement

This opcode allows defining a trigger for dynamic measurement.

Overview

`Opcode:` `0x30`

`Name:` `opcDT`

Function: Define Trigger

Parameter type: String 56

DLL-Function

MSC_WriteCommand 20

Request string to the Irinos-System

#{Trigger number};{Type};{Source};{Scaling factor};{Distance};{Start};{End}#

Trigger number

Number of the trigger. Two independent triggers can be defines. Permissible values are 1 and 2.

Type

T for Time-Trigger

P for Position-Trigger

Source

Position-triggered measurement: Name of the trigger channel (e.g. T7)

Time-triggered measurement: Unused. Use the character * as Source.

Scaling factor

Position-triggered measurement: Floating point value to convert the measurement value of the Trigger-Source into a physical unit, e.g. µm or °. This allows using physical units for the following parameters Distance, Start and End.
Use a scaling factor of 1 to use the measurement value unit (-> increments or digits).
The scaling factor can also be -1 in order to change the sign.

Time-triggered measurement: Unused. Always use the value 1.

Distance

Position-triggered measurement: Distance between 2 trigger-positions (floating point value). E.g. 1°.

Time-triggered measurement: Time period between two sample in ms, e.g. 0.1 for 100µs or 1 for 1ms.
The time-period must be a multiple of the sample periods of the Irinos-Boxes involved. The minimum value is 0.1 (=100 µs).
Most Irinos-Boxes have an internal sample period of 50µs. Thus valid values are for example 0.1, 0.25, 0.85, 1, 1.5, 12, etc.

Start

Position-triggered measurement: The measurement starts after this value has been exceeded (floating point value).

Time-triggered measurement: Delay until the start of the dynamic measurement in ms (floating point value).

End

Position-triggered measurement: The measurement will end, if this value is exceeded (floating point value). This also applied, if the maximum number of dynamic measurement values has not been reached yet.
Alternatively the character * can be used for this parameter. The dynamic measurement will then be active, until the maximum number of dynamic measurement values has been reached (or until it is stopped manually).

Time-triggered measurement: Duration of the dynamic measurement in ms (since start). It is recommended not to use an end-value by writing the character * into this parameter. A time-triggered measurement should be limited by the maximum number of measurement values as defined for the dynamic measurement (see opcDDMx 104).

Response string from the Irinos-System

`#0#`	Success
`#-n#`	Invalid parameter no `n`

`#-99#` General syntax error of the request string

Example parameter strings for position controlled measurement

---

```
#1;P;T2;20.0;0.1;50.0;*#
```

This string defines Trigger No. 1 as position-based. The position is taken from measurement channel T2, which could be for example the incremental encoder 2.

The binary measurement value (increments) must be divided by 20.0 to get the position value in mm.

The trigger shall occur 10 times / mm (every 0.1mm) and the first trigger-pulse shall be generated at the position 50.0mm (-> first values are sampled at 50.0mm).

No end is defined. Instead the character * tells the Irinos-System that the end-parameter is not used.

```
#2;P;T17;-1.0;10.0;0.0;3600.0#
```

This string defines Trigger No. 2 as position-based. The position is taken from measurement channel T17 (for example an incremental encoder with 3600 increments / rotation).

The measurement value of T17 shall not be converted to a physical unit. However, the measurement shall be performed into the negative direction. Therefore the sign is negated by the negative scale factor -1.0.

The trigger shall occur every 10 increments, which equals 1°-steps. The first pulse starts at 0 increments (= 0°). After 3600 increments (= 360°) the dynamic measurement shall be stopped.

Examples for time-triggered measurement

```
#2;T;*;1.0;1.0;0.0;*#
```

This string defines Trigger No. 2 as time-based. The sample period is 1.0ms, which equals 1 kSamples/s.

The dynamic measurement shall be started immediately (0.0ms).

Since no end value is used (character *), the dynamic measurement runs until the defined number of measurement values has been recorded (see opcDDMx) or until it is stopped manually via one of the opcodes opcIT[82] or opcDDMx[104].

```
#1;T;*;1.0;0.2;500.0;*#
```

This string defines Trigger No. 1 as time-based. The sample period is 0.2ms, which equals 5 kSamples/s.

The start of the dynamic measurement shall be delayed by 500.0ms.

Since no end value is used (character *), the dynamic measurement runs until the defined number of measurement values has been recorded (see opcDDMx[104]) or until it is stopped manually via one of the opcodes opcIT[82] or opcDDMx[104].

Comments

If a Position-Trigger is used, the values for scale, distance, start and end can be negative. For a Time-Trigger, these can only be positive.

It is possible to define an end-value for the trigger. The number of sampled measurement values is defined by the opcode opcDDMx[104]. If an end-point is defined, the dynamic measurement will be stopped automatically, if the end point is reached. This can lead to problems, if a defined number of samples is expected and the trigger has stopped the dynamic measurement before. In this case the defined number of samples will never be reached. Hence it is recommended to define either a end-value for a trigger OR a defined number of samples for the dynamic measurement.

A copy of the trigger configuration is made at the start of a dynamic measurement. If the trigger configuration is changed, this has no effect on an active dynamic measurement. It will become effective at the start of the next dynamic measurement.

Typically a separate trigger is used for each dynamic measurement (e.g. trigger 1 for dynamic measurement 1 and trigger 2 for dynamic measurement 2). Nevertheless, it is possible to use the same trigger for both dynamic measurements.

4.5.5 opcAT: Activate trigger for dynamic measurement

This opcode allows activating a trigger for a dynamic measurement. If the trigger is assigned to a dynamic measurement, this will start the dynamic measurement.

Overview

Opcode: 0x31

Name: opcAT

Function: Activate Trigger

Parameter type: String 56

DLL-Function

MSC_WriteCommand 20

Request string to the Irinos-System

#{Trigger number}#

Example: #2#

Trigger number

Number of the trigger, which shall be activated (1 or 2)

Response string from the Irinos-System

#0# Success

#-1# Invalid trigger number

#-99# General syntax error of the request string

Comments

If a dynamic measurement is started without an active trigger, no measurement values will be sampled.

In order to start a dynamic measurement, both, the measurement must be defined and activated (see opcDDMx[104]) and the trigger must be defined and activated (see opcDT[76] and opcAT[81]). It does not matter which of these two is defined and activated first.

4.5.6 opcIT: Inactivate trigger for dynamic measurement

This opcode allows inactivating a trigger for a dynamic measurement. If the trigger is assigned to a dynamic measurement, this will stop the dynamic measurement.

Overview

Opcode:	0x32
Name:	opcIT
Function:	Inactivate Trigger
Parameter type:	String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

#{Trigger number}#

Example: #2#

Trigger number

Number of the trigger, which shall be inactivated (1 or 2)

Response string from the Irinos-System

`#0#` Success

`#-1#` Invalid trigger number

`#-99#` General syntax error of the request string

4.5.7 opcSP: Set (channel-)parameter

This opcode allows parametrizing a measurement channel. The parameters depend on the channel type. Currently this opcode is only used for measurement channels for incremental encoders (1Vpp or TTL/RS422).

Overview

Opcode:	0x35
Name:	opcSP
Function:	Set (channel-)Parameter
Parameter type:	String 56

DLL-Function

MSC_WriteCommand 20

Request string to the Irinos-System

`#{Channel Name};{Position};{Reference index on/off}#`

Channel Name

Name of the measurement channel according to the channel assignment.

During startup all measurement channels are named in ascending order: T1, T2, T3, ...

The channel names can be changed via the channel assignment (opcWCA 69).

Position

- New position value for this measurement channel. This allows setting the position of the measurement channel.
- `*` if the position of the measurement channel shall not be changed. This is required, if only the reference index shall be turned on or off.
- `~` in order to reset the gain- and offset-control of the measurement channel. The position value will be reset to 0. This makes only sense for 1Vpp channels.
- `$` in order to completely reset the measurement channel. The incremental encoder will be switched off for about 10ms. The position will be set to 0.

Reference index on/off

Permissible parameter values:

- `REFON` to enable the reference index.
  In addition, the statusbit "Refmark" (see opcode opcRHS 86) will be enabled now, if the reference index is crossed.
- `REFOFF` to disable the reference index.

If the reference index is enabled, the position of the measurement channel will be set to 0, if the index is crossed.

Response string from the Irinos-System / measurement channel for incremental encoder (IR-INC)

`#0#` Success

`#-n#` Invalid parameter no `n`

`#-99#` General syntax error of the request string

Response string from the Irinos-System / measurement channel does not support this opcode

---

`#-98#`

Examples for request strings to incremental measurement channels

---

`#T5;-2000;REFOFF#`

The current position of the measurement channel T5 will be set to -2000. The reference index will be disabled.

`#T5;*;REFON#`

The reference index of the measurement channel T5 will be enabled. The position will not be changed.

`#T13;~;REFOFF#`

The gain- and offset control of the measurement channel T5 will be reset. The position will be set to 0. The reference index will be disabled.

Comments for incremental measurement channels

The error flags and the status bit "Refmark" will be cleared (these can be readout via the opcode opcRHS 86). Exception: This does not apply, if the character `*` is used for the parameter "position".

If the "position"-parameter $ is used, two of the measurement channels will be disabled for about 500ms. In this case, no position value will be available. This does also apply, if a dynamic measurement is active. Do not use this parameter for normal operation. It is intended for resetting a measurement

channel after a signal problems of the encoder have occurred. Depending on the selected measurement channels, the inputs 1 & 3 or 2 & 4 will be disabled.

Notes for 1Vpp measurement channels:

If an error is detected at the 1Vpp-inputs (see opcRHS 86), the signal levels of the incremental encoder are or have been out of specification. Further information can be found in the users manual of the Irinos-System.

After an error has occurred, it is recommended to reset the gain- and offset-control by using the character ~ as the position parameter.

A complete reset of the measurement channel via the position parameter $ is not required for the most applications.

After resetting the gain- and offset-control, it takes a few signal periods until the control has determined the optimal parameters. During this process, the interpolation accuracy is limited. The measurement values can be inaccurate (however, no increments are lost). Further, the signal tolerance is limited during this process.

4.5.8 opcRHS: Read hardware status of the measurement channels

This opcode allows reading the error status of each measurement channel. Further more for incremental channels, the reference index status bit 'Refmark' can be read out. It is set, if the reference index is enabled and crossed for the first time.

Overview

Opcode:	0x38
Name:	opcRHS
Function:	Read hardware status
Parameter type:	**Binary**

DLL-Function

The hardware status can be readout continuously via MSC_SetupStaticChannel[39] and MSC_ReadStatic[41]. This is the preferred way.

Alternatively it can be readout via MSC_WriteCommand[20].

Request data to the Irinos-System

```
Byte 0: Binary value 2
```

Response data from the Irinos-System

`Byte 0:`	Hardware status measurement channel 1
`Byte 1:`	Hardware status measurement channel 2
`Byte n:`	Hardware status measurement channel n+1

Status-Byte for measurement channels for incremental encoders

---

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PwrOvld		Refmark	Vector	GComp	OComp	AmpErr	Fast

PwrOvld

Error: A power supply overload of the incremental encoder(s) has been detected.

Refmark

The reference index has been crossed.

Vector

Error: The signal vector, which has been calculated from the cosine- and sine-signal, is too small. (Can only occur with 1Vpp incremental encoders.)

GComp

Error: The gain-control has reached its limit. (Can only occur with 1Vpp incremental encoders.)

OComp

Error: The offset-control has reached its limit. (Can only occur with 1Vpp incremental encoders.)

AmpErr

Error: One or both AD-converters for the measurement of the sine-/cosine-signal is/are overdriven. (Can only occur with 1Vpp incremental encoders.)

Fast

Error: The input frequency is too high.

Status-Byte for inductive probes Tesa or IET

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
							ShortCirc

ShortCirc

Error: Short circuit of the sine-oscillator

Status-Byte analogue inputs AIN

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
24VOvld	VRefOvld						

24VOvld

Error: Overload of the 24V output supply.

VRefOvld

Error: Overload of the reference voltage output.

Comments

The hardware status of the measurement channels should be checked regularly in order to verify the validity of the measurement values. Further details can be found in the users manual.

4.5.9 opcREv: Read current event of the Irinos-Boxes

Via this opcode the current event of the Irinos-Boxes can be readout. The same event will be displayed on the 7-digit display.

Overview

Opcode:	0x39
Name:	opcREv
Function:	Read event
Parameter type:	**Binary**

DLL-Function

MSC_WriteCommand 56

Request data to the Irinos-System

None

Response data from the Irinos-System

`Longword 0:` Current event of the first Irinos-Box

`Longword 1:` Current event of the second Irinos-Box

`Longword n:` Current event of the n-th Irinos-Box

(Longword = 32 Bit unsigned integer)

Comments

The numbers and descriptions of the events can be found in the Irinos users manual. 0 is "no event".

Events can be enabled or disabled using the opcode opcWEvCfg 92.

4.5.10 opcSAbsT: Set absolute time (for diagnostics memory)

This opcode allows setting the current date and time in the Irinos-System. This is not required but very useful for the diagnostic memory: date/time information will be added to all events.

Overview

`Opcode:`	`0x3A`
`Name:`	`opcSAbsT`
`Function:`	`Set absolute time`
`Parameter type:`	String 56

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

```
#{Value 1};{Year};{Month};{Day};{Hour};{Minute};{Second};
{Millisecond}#
```

Example: `#1;2015;06;26;16;49;32;532#`

Value 1

Value 1 must be used for this parameter.

Year

Year (4-digit)

Month

Month (1- or 2-digit)

Day

Day (1- or 2-digit)

Hour

Hour (1- or 2-digit) using the 24h time format

Minute

Minute (1- or 2-digit)

Second

Second (1- or 2-digit)

Millisecond

Millisecond (1-, 2- or 3-digit)

Response string from the Irinos-System

`#0#` Success

`#-n#` Invalid parameter no `n`

`#-99#` General syntax error of the request string

Comments

Writing the system-time is not required, but recommended. It provides date & time information for all diagnostic events.

After restarting the Irinos-System, the date/time information is lost. Therefore it should always be rewritten after power on.

The internal clock does not take into account leap years of leap seconds. Further it has no high accuracy. Hence it is recommended to rewrite the date/time information once a day. It can be rewritten any time.

4.5.11 opcWEvCfg: Write event configuration

This opcode allows changing the configuration of events, e.g. disabling the notification about an event.

For most applications there is no need to change the standard settings.

Overview

Opcode:	0x3D
Name:	opcWEvCfg
Function:	Write event configuration
Parameter type:	String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

```
#{No of Irinos-Box};{Value 1};{EventNo};{EventEnabled};{Max
number of diagnostic entries}#
```

Example: `#2;1;7;1;5#`

No of Irinos-Box

Number (address) of the Irinos-Box

Value 1

Value 1 must be used for this parameter.

EventNo

Event number. The available event numbers can be found in the users manual.

EventEnabled

- `0` --> Event is ignored.
- `1` --> Event is signalled, e.g. by displaying it on the 7-digit display.

Max number of diagnostic entries

This value is used for parametrizing the maximum number of entries in the diagnostic memory for the specific event since startup. By using the value 0, no diagnostic entries are written. The default value for most events is 10. This ensures that the diagnostic memory is not filled up with the same event multiple times, since this does usually not provide additional information.

Response string from the Irinos-System

`#0#` Success

`#-n#` Invalid parameter no `n`

`#-99#` General syntax error of the request string

Comments

Not all events can be parametrized. Further information can be found in the users manual.

The event configuration is only changed at the Irinos-Box, which is addressed by the parameter "No of Irinos-Box". If the event configuration shall be changed at multiple or all Irinos-Boxes, this configuration string must be send to all Irinos-Boxes. The same applies if multiple events shall be configured.

4.5.12 opcClrEv: Clear event

This opcode allows clearing (deleting) an active event.

Overview

Opcode:	0x3E
Name:	opcClrEv
Function:	Clear event
Parameter type:	String 56

DLL-Function

MSC_WriteCommand 20

Request string to the Irinos-System

`#{No of Irinos-Box};{Value 1};{EventNo}#`

Example: `#0;1;7#`

No of Irinos-Box

Number (address) of the Irinos-Box

Value 1

Value 1 must be used for this parameter.

EventNo

Event number. The available event numbers can be found in the users manual.

Response string from the Irinos-System

`#0#` Success

`#-n#` Invalid parameter no `n`

`#-99#` General syntax error of the request string

Comments

- Typically the event, which shall be cleared, has been readout using the opcode opcREv[89] before.
- If an event is cleared, which is not active, this opcode has no effect.
- Clearing an event does not eliminate the cause of the event! Depending on the event type, it may occur again immediately.
- The event configuration is only cleared at the Irinos-Box, which is addressed by the parameter "No of Irinos-Box". If the event configuration shall be cleared at multiple or all Irinos-Boxes, this configuration string must be send to all Irinos-Boxes.
- Some events are cleared automatically if the cause has been removed. This applies for example for the event "Sine oscillator short circuit", which is cleared as soon as the short circuit has been removed.

4.6 Opcodes: Measurement

4.6.1 opcRS: Read static measurement values

This opcode is used for reading all current (static) measurement values.

Overview

Opcode:	0x40
Name:	opcRS
Function:	Read static (measurement values)
Parameter type:	**Binary**

DLL-Function

The static measurement values can be readout continuously using the functions MSC_SetupStaticChannel[39] and MSC_ReadStatic[41]. This is the preferred way.

Alternatively they can be requested with MSC_WriteCommand[20].

Request data to the Irinos-System

None

Response data from the Irinos-System

Longword 0	Measurement value of measurement channel 1
Longword 1	Measurement value of measurement channel 2

`Longword n` Measurement value of measurement channel n+1

Comments

The longword is a 32 Bit signed integer. All measurement values are transferred as longwords. This applies also for measurement channels with 8 or 16 bit resolution.

The measurement channels are selected via the active channel list (see opcWCA[69], opcWCL[71] and opcACL[75]). After startup the channel list contains all measurement channels.

4.6.2 opcBIO: Read digital inputs / write digital outputs

This opcode is used for exchanging Bit I/O data (digital in- and outputs).

Overview

`Opcode:`	`0x42`
`Name:`	`opcBIO`
`Function:`	`Bit I/O`
`Parameter type:`	**`Binary`**

DLL-Funktion

The Bit I/O data can be exchanged continuously using MSC_SetupStaticChannel[39] and MSC_ReadStatic[41]. If output data (digital outputs) is changed, the function MSC_RefreshChannel[43] must be called (otherwise the MscDll won't use the updated output data). This is the preferred way.

Alternatively Bit I/O data can be exchanged using MSC_WriteCommand[20].

Request data to the Irinos-System

Byte 0	Digital outputs 1..8
Byte 1	Digital outputs 9..16
Byte 2	Digital outputs 17..24
etc.	

Response data from the Irinos-System

Byte 0	State of the digital outputs 1..8
Byte 1	State of the digital outputs 9..16
Byte 2	State of the digital outputs 17..24
etc.	
Byte x+0	State of the digital inputs 1..8
Byte x+1	State of the digital inputs 9..16
Byte x+2	State of the digital inputs 17..24
etc.	

Comments

The response data first contains "mirrored" output bytes. The data length of the digital outputs is always identical to the data length of the digital inputs. Examples:

- If 8 output bytes are written to the request data, the response contains 8 (mirrored) bytes for the output bytes plus 8 additional bytes for the input data.
- If 16 output bytes are written to the request data, the response contains 16 (mirrored) bytes for the output bytes plus 16 additional bytes for the input data.

The number of in-/output data bytes is independent of the real number of digital in- and outputs available:

- If more output bits are written than digital outputs are available, the additional bits will be ignored.
- If less input bits are read than digital inputs are available, the status of the additional inputs is not transferred to the PC.
- If more input bits are read than digital inputs are available, all additional input bits read as 0.

Exchanging Bit I/O data is quite similar to reading static measurement values. Typically the data is exchanged between the MscDll and the Irinos-System continuously. An important difference is, that Bit I/O required a bidirectional data transfer.

Usually this opcode is used together with the function MSC_SetupStaticChannel[39]. This function defines a send- and a receive-buffer. The send-buffer contains the output bits and the receive-buffer the inputs bits.

The MscDll contains a local buffer for the output data. If the application wants to change the output data, it must tell this the MscDll via the function MSC_RefreshChannel[43].

4.6.3 opcBIORO: Read current state of digital in-/outputs

This opcode is available from firmware version 1.4.x.x.

Via this opcode the current state of the digital in-/outputs can be retrieved.

It uses the same data format as the opcode opcBIO[97], except that the new output states are not applied (-> read only). It is typically used after (re-)starting the measurement software in order to get the current in-/output state without changing the outputs.

Overview

Opcode:	0x43

Name: opcBIORO

Function: Bit I/O

Parameter type: **Binary**

DLL-Funktion

MSC_WriteCommand[20]

Request data to the Irinos-System

Byte 0 Digital outputs 1..8

Byte 1 Digital outputs 9..16

Byte 2 Digital outputs 17..24

etc.

In order to have the same data format as the opcode opcBIO[97], output data must be written here. However, the state of the outputs will not be changed (-> read only).

Response data from the Irinos-System

Byte 0 State of the digital outputs 1..8

Byte 1 State of the digital outputs 9..16

Byte 2 State of the digital outputs 17..24

etc.

Byte x+0 State of the digital inputs 1..8

Byte x+1 State of the digital inputs 9..16

Byte x+2 State of the digital inputs 17..24

etc.

Comments

The response data first contains "mirrored" output bytes. The data length of the digital outputs is always identical to the data length of the digital inputs. Examples:

- If 8 output bytes are written to the request data, the response contains 8 (mirrored) bytes for the output bytes plus 8 additional bytes for the input data.
- If 16 output bytes are written to the request data, the response contains 16 (mirrored) bytes for the output bytes plus 16 additional bytes for the input data.

The number of in-/output data bytes is independent of the real number of digital in- and outputs available:

- If less input bits are read than digital inputs are available, the status of the additional inputs is not transferred to the PC.
- If more input bits are read than digital inputs are available, all additional input bits read as 0.

4.6.4 opcRSW: Read statusword for dynamic measurement

This opcode allows reading the statusword for dynamic measurement.

Overview

Opcode:	0x44
Name:	opcRSW
Function:	Read statusword (for dynamic measurement)
Parameter type:	**Binary**

DLL-Function

MSC_WriteCommand[20]

Request data to the Irinos-System

None

Response data from the Irinos-System

Longword (32 Bit) with status information. See table below.

Comments

A status-change of the following bits may be delayed:

- Trigger X was active and is now inactive
- Trigger X: Min 1 trigger pulse occurred
- Dynamic measurement X was active and is now inactive
- Dynamic measurement X: Min 1 value has already been sampled

Statusword for dynamic measurement

Bit No.	Function
0	Trigger 1 active
1	Trigger 1 was active and is now inactive
2	Trigger 1: Min 1 trigger pulse occurred

Bit No.	Function
3	
4	Dynamic measurement 1 is active
5	Dynamic measurement 1 was active and is now inactive
6	Dynamic measurement 1: Min 1 value has already been sampled
7	Dynamic measurement 1: Reading values by the PC is active
8	Dynamic measurement 1: Internal sample buffer is full
9	
10	
11	
12	
13	
14	
15	
16	Trigger 2 active
17	Trigger 2 was active and is now inactive
18	Trigger 2: Min 1 trigger pulse occurred
19	
20	Dynamic measurement 2 is active
21	Dynamic measurement 2 was active and is now inactive
22	Dynamic measurement 2: Min 1 value has already been sampled
23	Dynamic measurement 2: Reading values by the PC is active
24	Dynamic measurement 2: Internal sample buffer is full

Bit No.	Function
25	
26	
27	
28	
29	
30	
31	

4.6.5 opcDDMx: Define dynamic measurement

A dynamic measurement is configured via this opcode.

Overview

```
Opcode:           0x50 for dynamic measurement 1
                  0x51 for dynamic measurement 2

Name:             opcDDM1 for dynamic measurement 1
                  opcDDM2 for dynamic measurement 2

Function:         Define dynamic measurement

Parameter type:   String
```

DLL-Function

MSC_WriteCommand

Request string to the Irinos-System

```
#{Trigger number};{Number of channel list};{Active};{Max.
number of samples}#
```

Trigger number

Number of the trigger (1 or 2), which is used for the dynamic measurement. It must be defined using the opcode opcDT[76].

Number of channel list

Number of the channel list (1 .. 10), which shall be used for the dynamic measurement. It must be defined using the opcode opcWCL[71].

Active

1 to enable the dynamic measurement.
0 to disable the dynamic measurement.

Max. number of samples

Max number of samples to be stored. Use * for ‚no limit'.

Response string from the Irinos-System

```
#0#   Success

#-n#  Invalid parameter no n

#-99# General syntax error of the request string
```

Comments

In order to start a dynamic measurement, both, the measurement must be defined and activated and the trigger must be defined and activated (see opcDT[76] and opcAT[104]). It does not matter which of these two is defined and activated first.

There are 4 possibilities to stop a dynamic measurement:

- Via the parameter "active = 0" of this opcode.
- Via a "max number of samples" of this opcode. The measurement will stop if the number of samples have been stored.

- By inactivating the assigned trigger (see opcIT[82]).

- By defining an end value for the trigger (see opcDT[76]).

An endless measurement is not allowed. The typical duration of a dynamic measurement is < 60s.

A dynamic measurement samples all measurement channels, which are contained in the assigned channel list. This results in a curve for each measurement channel. For each curve a buffer must be provided using the function MSC_AttachSubChannelBuffer[46]. It is very important, that all required buffers are provided and that they have an appropriate size.

The channel list 0 always contains all measurement channels and should not be used for the dynamic measurement.

4.6.6 opcRDMx: Read dynamic measurement values

This opcode is required for reading dynamic measurement values, which have been stored in the Irinos-System.

Overview

Opcode:	0x60 for dynamic measurement 1 0x61 for dynamic measurement 2
Name:	opcRDM1 for dynamic measurement 1 opcRDM2 for dynamic measurement 2
Function:	Read dynamic measurement (values)
Parameter type:	**Binary**

DLL-Function

Reading the dynamic measurement values is a complex process. The interpretation of the binary data is always done by the MscDll.

In order to read dynamic measurement values, a dynamic transfer channel must be used:

- A dynamic transfer channel is established using MSC_SetupExtendedDynamicChannel[44] .
- For each measurement channel used in a dynamic measurement, a buffer must be allocated. This buffer must be assigned to the MscDll using the function MSC_AttachSubChannelBuffer[46].

The MscDll automatically writes the measurement values into the assigned buffers. The application can read directly from these buffers. All measurement values are stored as 32 bi signed integer values.

During and after a dynamic measurement, the function MSC_GetPosition[49] allows retrieving the number of measurement values, which have already been stored in the buffers (--> filling level of the buffers). All buffers of the same dynamic measurement have the same filling level.

4.7 Opcodes: Service

4.7.1 opcRST: System Reset

This opcode allows restarting the whole Irinos-System. Only use this opcode after consulting the manufacturer.

Overview

Opcode:	0x7E
Name:	opcRST
Function:	System-Reset
Parameter type:	String[56]

DLL-Function

MSC_WriteCommand[20]

Request string to the Irinos-System

```
#RESET_MTS;{Delay Master-Box};{Delay Slave-Boxen}#
```

Delay Master-Box

Time in ms until the Master-Box (-> first Irinos-Box) will be restarted.

Delay Slave-Boxen

Time in ms until all Slave-Boxes will be restarted.

Response string from the Irinos-System

```
#0#   Success

#-n#  Invalid parameter no n

#-99# General syntax error of the request string
```

Comments

The delay for the Master must be longer than the delay for the Slaves. Use the following values:

```
#RESET_MTS;2000;500#
```

The reset command is send to all Slave-Boxes via the ILink interface. If the ILink interface does not work properly, no slave can be reset.

- 1 -

- 2 -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- V -

- W -