

	<p>Commands of the qs-STAT Statistical Server</p>	<p>Page 1 / 81</p>
---	---	--------------------

Commands of the qs-STAT Statistical Server

1	Using the qs-STAT COM Interface	5
1.1	Remote Control of the Server	5
1.2	Connection Points	5
1.3	Enumerating supported Functionalities	5
1.4	General remarks	7
1.4.1	XxxaVA methods / untyped [out] parameters.....	7
1.4.2	XxxxTM methods / concurrent processing	7
2	TQSSTATREMOTECONTROL	8
2.1	Getting started.....	9
2.1.1	Connecting to the Server	9
2.1.2	Exiting the Session.....	10
2.2	Loading data to the Server	11
2.2.1	Opening an existing file	11
2.2.2	Sending data in a stream	12
2.2.3	Querying the Database.....	15
2.2.4	Create database queries.....	15
2.3	Information about loaded data.....	16
2.3.1	GetGlobalInfo.....	16
2.3.2	Informations about loaded parts.....	17
2.3.3	Informations about loaded characteristic	17
2.3.4	Informations about loaded values	18
2.4	Getting results	19
2.4.1	Adressing characteristics	19
2.4.2	Evaluation the data.....	19
2.4.3	excluding characteristics from the statistic	21
2.4.4	Getting graphical results	22
2.4.5	Getting or printing predefined reports.....	25
2.4.6	Getting numerical results	29
2.5	Navigating in the displayed graphics.....	32
2.5.1	Decode the cursor position.....	32
2.5.2	Mark a data position	35
3	Accessing the Q-DAS Database.....	38
3.1.1	GetFirstQueryName.....	38
3.1.2	GetNextQueryName.....	39
3.1.3	LoadQuery.....	39
3.1.4	CreateQuery.....	40
3.1.5	AddFilterToQuery.....	40

3.1.6	AddSortToQuery	41
3.1.7	AddFunctionToQuery	42
3.1.8	AddAutoSelectToQuery.....	42
3.1.9	SetQueryProperty	42
3.1.10	SaveQuery	44
3.1.11	ExecuteQuery	44
3.1.12	GetFirstPartQuery.....	45
3.1.13	GetNextPartQuery.....	46
3.1.14	SkipPartQuery.....	47
3.1.15	GetFirstCharQuery.....	47
3.1.16	GetNextCharQuery.....	48
3.1.17	SkipCharQuery.....	49
3.1.18	GetFirstValueQuery.....	49
3.1.19	GetNextValueQuery.....	50
3.1.20	SkipValueQuery.....	51
3.1.21	FreeQuery	51
3.1.22	GetFirstFilterName.....	52
3.1.23	GetNextFilterName.....	52
3.1.24	LoadFilter.....	53
3.1.25	CreateFilterFromFilters.....	53
3.1.26	CreateFilter.....	54
3.1.27	CreateFilterFromSQL	55
3.1.28	SaveFilter	55
3.1.29	FreeFilter	56
3.1.30	CreateDirectSQL	56
3.1.31	ExecuteDirectSQL.....	57
3.1.32	GetFirstDirectSQLRow	58
3.1.33	GetNextDirectSQLRow	58
3.1.34	FreeDirectSQL.....	59
3.1.35	RecentSerNo_First.....	59
3.1.36	RecentSerNo_Next	60
3.1.37	Get_n_Buildphases	60
3.1.38	GetBuildphases	61
3.1.39	Benchmark.....	62
3.1.40	OpeningLine2	62
3.2	Events	64
3.2.1	Alive State.....	64
3.2.2	Server Intialization	65
3.2.3	Data loading Synchronization	65
3.2.4	Statistical Evaluation Synchronisation	66

3.3	Alarms	67
3.3.1	OnSystemAlarm.....	67
3.3.2	OnStatisticalAlarm	68
4	Additional Interfaces	70
4.1	Interface IEnumLanguage	70
4.2	Interface IEnumModule.....	71
4.3	IEnumEvaluationStrategy	72
4.3.1	Dependencies	72
4.4	Interface IEnumGraphic	73
4.4.1	Dependencies	73
4.5	Interface IEnumStatResults	73
4.5.1	Dependencies	73
4.6	Interface IEnumReports	73
4.6.1	Dependencies	73
4.7	Interface IEnumCatalogue.....	74
4.8	Interface IEnumFieldKeyName	74
4.9	Interface IEnumDBSelectionName.....	75
4.10	Interface IQsstatData	76
4.10.1	SetKey.....	76
4.10.2	SortValues	77
4.10.3	DeleteValues	78
4.10.4	DFDCharNr2PartCharNr.....	78
4.10.5	PartCharNr2DFDCharNr.....	79
5	DCOM Security Aspects	81

1 Using the qs-STAT COM Interface

The Q-DAS Server implements currently the following interfaces:

1.1 Remote Control of the Server

- ITQsstatRemoteControl
(uuid(BF95CF2B-954B-11D4-B1A1-00105AD88C31))
general access to qs-STAT functionalities
- IDataBase
(uuid(11853321-12CB-11D5-B22D-000105AD88C31))
extended access to the qs-STAT database, possibilities to create queries on client side

1.2 Connection Points

- IQsEvents
(uuid (BF95CF2C-954B-11D4-B1A1-00105AD88C31))
Event handling by connection points

1.3 Enumerating supported Functionalities

- IEnumModule
(uuid(7D9DF503-AF0E-11D4-B1AE-00105AD88C31))
getting all available modules
- IEnumLanguage
(uuid(7D9DF501-AF0E-11D4-B1AE-00105AD88C31))
getting all available languages
- IEnumEvaluationStrategy
(uuid(7284B001-8733-11D5-A75B-00105A27BB6))
getting all defined evaluation strategies
- IEnumGraphic
(uuid(7D9DF507-AF0E-11D4-B1AE-00105AD88C31))
getting all available graphics
- IEnumStatResults
(uuid(7D9DF505-AF0E-11D4-B1AE-00105AD88C31))
getting all available statistical calculations

- IEnumReports
(uuid(7D9DF509-AF0E-11D4-B1AE-00105AD88C31))
getting a list of predefined reports
- IEnumCatalogues
(uuid(08D25F64-8D44-4C5B-AD0A-8EE723457006))
retrieve a list of all loaded / activated catalogues and catalogue entries
- IEnumFieldKeyName
uuid(BF95CF2E-954B-11D4-B1A1-00105AD88C31))
getting all available K-Fields
- IEnumDBSelectionName
(uuid(BF95CF2D-954B-11D4-B1A1-00105AD88C31))
Getting all the available database queries

1.4 General remarks

The interface TQSSTATREMOTECONTROL defines some methods, which are named as xxxxVA, xxxxTM, where xxxx is the name of the general methods. They were designed especially for Scripting languages, which are not able to handle typed variables or event handling.

1.4.1 XxxaVA methods / untyped [out] parameters

All of the [OUT] parameters are typed as variant, which have the same meaning as the [OUT] parameters of the related method without VA-extension.

1.4.2 XxxxTM methods / concurrent processing

These methods can be used instead of the concurrently processed methods for loading data or evaluation data. They have an additional parameter named ThreadMode. This parameter can be used to switch the concurrently processing of

Meaning of the parameter ThreadMode:

ThreadMode	
0	Command is processed concurrently, events will be signaled
1	Command is processed sequentially, events wont be signaled

1.4.3 Addressing parts and characteristics

Methods often have parameters for addressing parts and characteristics. They are always used in a combination and are in general declared either as INTEGER or VARIANT values.

Basic rules:

- Parts are numbered relative to the connection handle
- Characteristics are numbered relative to the PartNr parameter

If declared as an INTEGER value, the meaning is the following:

- PartNr and CharNr \neq 0: a certain characteristic is addressed. The number (position) of the characteristic is relative to the part.
- PartNr and CharNr = 0: all characteristics present at a given connection handle are addressed (if functionality is possible).
- PartNr \neq 0 and CharNr = 0: all characteristics of a certain Part are addressed

If declared as a VARIANT value, the meaning is the following:

- either integer values can be passed, in that case the rules correspond to the rules mentioned before
- in addition 1 dimensional arrays of integer values can be passed. In that case a list of characteristics is addressed (if possible). Both arrays for PartNr and CharNr parameters have to be set and filled to the same size

Example: To Access char Nr 1, 3, 5 of the same Part, the list should look like the following:

index	PartNr array	CharNr array
1	1	1
2	1	3
3	1	5

2 TQSSTATREMOTECONTROL

2.1 Getting started

After creating the COM object by using the `CoInitializeEx(NULL, COINIT_MULTITHREADED)` and `CoCreateInstance` with the parameters `CLSCTX_LOCAL_SERVER` or `CLSCTX_REMOTE_SERVER`. The client application has to register itself by using the command: `ClientConnect`.

The client gets a handle, that has to be stored inside the client and be given to the interface at each command.

2.1.1 Connecting to the Server

This method creates a new data instance for the client in the requested module and loads the settings for a certain user. The returned handle has to be stored on the client side to have later an access to the created data instance. This method checks the registration codes for module and language. It also checks the user and the password according to the qs-STAT user administration.

If the method succeeds the server returns a succeeded HRESULT value.

Reasons for a denial of access could be:

- invalid or not registered module
- invalid or not registered language
- unknown user
- invalid user password

2.1.1.1 IDL Syntax

```
[id(0x00000016)]
HRESULT _stdcall ClientConnect(      [out] long * handle,
                                     [in] long ModuleID,
                                     [in] long LanguageID,
                                     [in] BSTR UserID,
                                     [in] BSTR UserPwd );
```

2.1.1.2 Parameters

Handle	This command gives a handle back to the Client application, that has to be stored and given to the server at every command.
ModuleId	Key for the startup module for qs-STAT: - 1: reliability analysis - 10: sample analysis

	- 20 process capability - 26 procella (Online Server / Online Analysis Server) - 30 measurement system analysis
LanguageId	Country code for the startup language (e.g. 1 = English, 49 = German)
UserPwd	Password of the user

NOTE:

The ID's for the qs-STAT modules can be retrieved by the IEnumModule-interface. Several connections can be established at run time. Each connection is represented by a unique Handle, which is retrieved by a call to the ClientConnect methods. The Client application has to keep track of the connections itself at least by storing the retrieved handles.

2.1.2 Exiting the Session

For a correct logoff of the client, it's recommended to call the disconnect procedure, which removes the client from the internal server structures and releases the allocated memory. The function is named `ClientDisconnect`. This command logs the clients off.

2.1.2.1 IDL Syntax

```
[id(0x00000016)]
HRESULT _stdcall ClientDisconnect( [in] long handle);
```

2.1.2.2 Parameters

Handle	Qs-STAT handle got by the client connect command
--------	--

2.2 Loading data to the Server

After the connect is done, it's necessary to load data to the server or getting system information for configuration purpose. Please note, that without loaded data no statistical results will be available. This chapter describes the ways how to load data to the server. This loaded data may be evaluated later.

The client has 3 possibilities to load data inside the server:

1. Open an existing file
2. Sending the data via stream
3. Querying the Q-DAS database
4. Create database queries on Client side

These commands are processed concurrently inside the server. It's recommended that the client reacts internally to the event `OnDataAvail` defined inside the `IqsEvents` interface.

After the data is loaded, informations about the loaded data can be retrieved by the methods

- `GetGlobalInfo` (number of loaded parts, characteristics, values)
- `GetPartInfo` (retrieve informations about a certain part (contents of K-Fields K1xxx))
- `GetCharInfo` (retrieve informations about a certain characteristics (contents of K-Fields K2xxx, K8xxx))
- `GetSingleValueExt` (retrieve informations about measured values (contents of K-Fields K0xxx))

2.2.1 Opening an existing file

The client may call the `OpenFile` command to load a specific file in the Q-DAS data format. Probably this is the easiest and most efficient way, if you are dealing with a local server. It's important to know, that both, the client and the server are using and understanding the same drive and directory names. This cannot be guaranteed using a remote server.

Note: This command is processed concurrently, so a synchronisation is recommended. This command should only be used, if client and server run on the same machine. Otherwise the `TransmitFileXXX` methods have to be used.

2.2.1.1 IDL Syntax

```
[id(0x00000001)]
HRESULT _stdcall OpenFile(  [in]    long   handle,
                           [in]    BSTR   filename);
```

2.2.1.2 Parameters

Handle	Qs-STAT handle got by the client connect command
Filename	Name of the file to be loaded by the server

2.2.1.3 Events

After this command is processed by the server, the Event `OnDataAvail` is called. The number of characteristics loaded is given to the client through this event (See `IQsEvents`).

Remarks:

- 1.) The file consists of Characters formatted according to the specifications of the Q-DAS Ascii-Transferformat.
- 2.) Online Server: The data is evaluated immediately without a call to `EvaluateXXXX` – methods and the alarm events will raised if occurred.

2.2.2 Sending data in a stream

Alternatively the client may send data in form of a stream to the server. This avoids the file handling and may be useful especially when running the server on a remote machine. The transmission may be split in several smaller portions.

The commands are:

- `TransmitFile` for transmitting the data in one stream.
- `TransmitPartFile` for transmitting the data in portions
- `TransmitFileExt` for Loading new data and / or appending data to previously loaded data sets

If the client wants to transmit the file in portions, he has to implement the `IQSEvent` interface. The events

```
[id(0x00000002)]
HRESULT OnDataAvail([in] long NumOfChars );
[id(0x00000009)]
HRESULT OnDataAvailExt([in] long handle, [in] long NumOfChars );
```

are called after the server has processed one portion and also when the transmission is finished.

Note: These commands are processed concurrently, so a synchronisation is recommended.

2.2.2.1 IDL Syntax

```
[id(0x00000013)]
HRESULT _stdcall TransmitFileExt(  [in] long      handle,
                                   [in] VARIANT  packet
                                   [in] mode      unsigned char;
                                   [in] reserve  long);
```

2.2.2.2 Parameters

Handle	Qs-STAT handle got by the client connect command
Packet	Memory stream of the data in the Q-DAS data format (array of Char).
Mode	decides, how new data has to be handled, see table below
Reserve	reserve for future development

Mode:

Value	Function
0	Resets the previously loaded data and creates new data structures for the data inside the stream (create new parts, characteristics and add the values to the newly created parts and characteristics)
2	Keeps the previously loaded data and tries to append the values to the existing data set. Ignores existing part and characteristic data
6	Appends values initially (in case of DFD / DFX usage to append the first stream of DFX data). See also mode = 2. This mode effects the Online Server / Online Analysis Server only.
8	Switches from relative to absolute addressing mode for measured values and additional data inside the stream.

Reserve:

Bitcoded parameter for external control

Value	Function
0x00FF	Reserved
0x0100	Used in module 26 ("Online Server"). Skip the automatic evaluation in case of multiple sequential calls of transmitfileExt (when transmitting larger amount of data in several packages)
0x0200	Perform a calculation of those characteristics, which have a logical formula defined

0x0400	Used in module 26 ("Online Server"). Skips the automatic QCC usage for the Online QCC / alarm QCC and forces the program to use a QCC, which was previously transmitted for the characteristic
--------	--

2.2.2.3 Events

After this command is processed by the server, the Event `OnDataAvail` and `OnDataAvailExt` are called. The number of characteristics loaded is given to the client through this event (See `IQsEvents`).

When using module 26 (Online Server) depending on the transmitted data and the presetted evaluation strategy (see `SetEvaluationStrategy`) the Event `OnStatisticalAlarm` and `OnStatisticalAlarmExt` can be raised (See `IQsEvents`).

Remarks:

- 1.) The stream consists of an array of Characters formatted according to the specifications of the Q-DAS Ascii-Transferformat.
- 2.) Online Server: The data is evaluated immediately without a call to `EvaluateXXXX` – methods and the alarm events will be raised if occurred. Can be skipped by using the reserve parameter.

2.2.3 Querying the Database

If the client knows the names of the stored database queries for the current user, the client may call one query to load the data according to the requested query from the Q-DAS database. To get the available queries, see `IEnumDBSelectionName` interface.

Note: This command is processed concurrently, so a synchronisation is recommended.

For more detailed possibilities for filtering and querying the database, see the `IDatabase` interface.

2.2.3.1 IDL Syntax

```
[id(0x00000006)]
HRESULT _stdcall DBOpen(      [in]    long    handle,
                               [in]    BSTR    szlSelektionsname );
```

2.2.3.2 Parameters

Handle	Qs-STAT handle got by the client connect command
SzlSelektionsname	Name of the specific database query

2.2.3.3 Events

After this command is processed by the server, the Events `OnDataAvail` and `OnDataAvailExt` are called. The number of characteristics loaded is given to the client through this event (See `IQsEvents`).

2.2.4 Create database queries

For details see the `IDatabase` interface. This allows a client application to query data of all levels of the Q-DAS data model.

2.3 Information about loaded data

After a data set is loaded in the server, informations about the loaded data can be retrieved. The function is named `GetGlobalInfo`

This command returns information about the number of loaded parts, number of loaded characteristics, number of loaded measurement values.

2.3.1 GetGlobalInfo

2.3.1.1 IDL Syntax

```
[id(0x0000000f)]
HRESULT _stdcall GetGlobalInfo(      [in] long   handle;
                                     [in] long   part_nr;
                                     [in] long   char_nr;
                                     [in] long   Op_code;
                                     [out] long* ret);
```

2.3.1.2 Parameters

Handle	Qs-STAT handle got by the client connect command
Part_nr	Number of the part of the loaded data to retrieve informations
Char_Nr	number of the characteristic of the loaded to retrieve information about
Op_code	kind of desired information:
Ret	Return value: content depends on the <code>Op_code</code> parameter

Op_code:

Op_code	Return value
General Information about the loaded data	
1	Total number of loaded parts
2	Total number of characteristics
3	Number of characteristics for a certain part (referenced by <code>part_nr</code>)
4	Number of values for a certain part/characteristic (referenced by <code>part_nr</code> , <code>char_nr</code>)
System informations	
101	FIFO size (Online Server only)
111	Total number of evaluated samples (Online Server only). Depending on the selected evaluation strategy. This is also the default for the displayed samples in requested graphics.

2.3.2 Informations about loaded parts

This functions returns informations for the attributes of a loaded and requested part. The attributes are coded according to the Q-DAS data format description.

2.3.2.1 IDL Syntax

```
[id(0x0000000D)]
HRESULT _stdcall GetPartInfo( [in] long handle,
                              [in] long KFieldNr,
                              [in] long Part_Nr,
                              [in] long Char_Nr,
                              [out] BSTR * KFieldValue );
```

2.3.2.2 Parameters

Handle	Qs-STAT handle got by the client connect command
KFieldNr	Number of the K-Field to get informations about (K2xxx, K8xxx)
Part_nr	internal number of the part of the loaded data
Char_Nr	not used, should be set to 0
Value_Nr	internal number of the value
KFieldValue	return value: content of the K-Field (KfieldNr) for the requested part (part_nr)

2.3.3 Informations about loaded characteristic

This functions returns informations for the attributes of a loaded and requested characteristic. The attributes are coded according to the Q-DAS data format description. The characteristic has to be referenced relatively to the part and depends on the state of the characteristic (selected, deselected)

2.3.3.1 IDL Syntax

```
[id(0x0000000D)]
HRESULT _stdcall GetCharInfo( [in] long handle,
                              [in] long KFieldNr,
                              [in] long Part_Nr,
                              [in] long Char_Nr,
                              [out] BSTR * KFieldValue );
```

2.3.3.2 Parameters

Handle	Qs-STAT handle got by the client connect command
KFieldNr	Number of the K-Field to get informations about (K2xxx, K8xxx)
Part_nr	internal number of the part of the loaded data
Char_Nr	internal number of the characteristic (relative to the part)

KFieldValue	return value: content of the K-Field (KfieldNr) for the requested characteristic (referenced by part_nr, char_nr)
-------------	---

2.3.4 Informations about loaded values

This functions returns informations for the attributes of a loaded and requested value (including additional data).The attributes are coded according to the QDAS data format description.

2.3.4.1 IDL Syntax

```
[id(0x0000000D)]
HRESULT _stdcall GetValueInfo(
    [in] long handle,
    [in] long KFieldNr,
    [in] long Part_Nr,
    [in] long Char_Nr,
    [in] long Value_Nr,
    [in] unsigned char bTransformation,
    [out] BSTR * Value );
```

2.3.4.2 Parameters

Handle	Qs-STAT handle got by the client connect command
KFieldNr	Number of the K-Field to get informations about (K0001 .. K0053)
Part_nr	internal number of the part of the loaded data
Char_Nr	internal number of the characteristic (relative to the part)
Value_Nr	internal number of the value
BTransformation	for later developments, should be set to 0
Value	Return valu: content of the K-Field (KfieldNr) for the requested value (referenced by part_nr, char_nr, value_nr)

2.4 Getting results

After data is load to the server, the client should call an statistical evaluation of the data. After this evaluation is done, the results are ready to be called by a client application.

2.4.1 Adressing characteristics

The Q-DAS server has a unified addressing mode for parts and characteristics. The view onto the data is independent from the data source, either if the data source is a file/stream or database. The enumeration of the parts starts new at each connection established to the server. The enumeration of charaterstic starts new at each part. When using higher – leveled characteristic structures (grouped characteristics) the enumeration is descending the characteristic tree. Especially at file / stream based data the order of the characteristic might change, when this feature is used.

These logical characteristic numbers (the combination of Part number and characteristic number) have to be used to address characterstics for getting results.

2.4.2 Evaluation the data

These two methods below perform a statistical evaluation of the loaded data set according to the presetted qs-STAT module and according to the predefined and selected evaluation method.

The client has the possibility to

- evaluate all loaded characteristics
- evaluate a specific characteristic

These commands are processed concurrently, so a synchronisation by using the `OnEvaluationDoneXXX` events is recommended. If a concurrent processing isn't wanted, the `EvaluateXXXXTM` methods can be used (`ThreadMode = 1`).

2.4.2.1 IDL Syntax

```
[id(0x00000002)]
HRESULT _stdcall EvaluateAllChars( [in] long handle );
[id(0x00000004)]
HRESULT _stdcall EvaluateChar(      [in] long handle,
                                   [in] long part_Nr,
                                   [in] long char_Nr);
HRESULT _stdcall EvaluateCharExt(  [in] long handle,
                                   [in] VARIANT_BOOL SPCEvaluation,
                                   [in] int threadmode,
                                   [in] long part_Nr,
                                   [in] long char_Nr,
                                   [in] long reserve1,
                                   [in] long reserve2);
```

2.4.2.2 Parameters

Handle	Qs-STAT handle got by the client connect command
Part_Nr , Char_Nr	To Address a certain characteristic of the loaded data, Char_nr is relative to the part
ThreadMode	Indicates, wether the command should be processed concurrently (0) or sequentially (1)
SPCEvaluation	Used only in module 26 (Online Analysis Server). True indicates, that a process capability evaluation based on the whole range of the values FIFO should be done or if the general evaluation for module 26 should be performed (which is generally / automatically done, when data has been transmitted or changed).
Reserve1	For external control, see table below
Reserve2	Reserved for future developments

Reserve1:

Bitcoded parameter for partially control of the evaluations behaviour

value	Effect
0x0001	Used at module 26 ("Online analysis Server") in order to skip the automatic takeover for the Online-QCC / Alarm QCC completely.

2.4.2.3 Events

After one of these commands are processed by the server, the Events
OnEvaluationDone and OnEvaluationDoneExt is called (See IQsEvents).

2.4.3 excluding characteristics from the statistic

For some reasons it is helpful to exclude certain characteristics from the statistic. Therefore the command `ToggleChar` can be used to exclude or include certain characteristics.

2.4.3.1 IDL Syntax

```
[id(0x00000005)]
HRESULT _stdcall ToggleChar( [in]          long      handle,
                             [in, out]    long *    Part_Nr,
                             [in, out]    long *    Char_Nr,
                             [in]         long      Toggle );
```

2.4.3.2 Parameters

Handle	Qs-STAT handle got by the client connect command
Part_Nr, Char_Nr	To Address a certain characteristic of the loaded data
Toggle	decides what to do with the addressed characteristic (select, deselect, etc.), see table below

Toggle:

Value	Meaning
0	Toggles the characteristics (exclude, if characteristic is currently included, include if characteristic is currently excluded)
1	Include the addressed characteristic to the statistic / graphic
2	Exclude the addressed characteristic from the statistic / graphic

Remarks:

Please note that basically the startup status is defined in the K-field K2080 inside the data set. If not set inside the data set, the characteristic is automatically included. After data is loaded in the server, the characteristic can be changed by the `ToggleChar` command. The result of this command isn't stored inside the data set itself, it is available at run time only. Storing this information persistently is task for the client application.

2.4.4 Getting graphical results

By using this method the client can get the available graphics of the server. Each specific graphic inside the server has a unique number (key). The server generates a graphic in the desired size and passing it through the interface in form of a stream. Currently the graphic formats are bitmap, JPEG, Metafile, enhanced metafile. The available graphics and their keys (see Graphic_nr) maybe retrieved through the `IEnumGraphic` interface.

2.4.4.1 GetGraphic

A very easy way to get a graphic in a bitmap format.

2.4.4.1.1 IDL Syntax

```
[id(0x00000010)]
HRESULT _stdcall GetGraphic( [in] long      handle,
                             [in] long      Graphic_Nr,
                             [in] long      part_Nr,
                             [in] long      char_Nr,
                             [in] long      width,
                             [in] long      height,
                             [out] VARIANT* bmp );
```

2.4.4.1.2 Parameters

Handle	Qs-STAT handle got by the client connect command
Graphic_Nr	Unique key for a certain desired graphic (See <code>IEnumGraphics</code>)
part_Nr	Internal number of the certain part of the loaded data, for that the the graphic shall be generated.
char_Nr	Internal number of the specific characteristic of the loaded data, for that the the graphic shall be generated.
width	Desired width of the graphic (dimension) in pixel
Height	Desired height of the graphic (dimension) in pixel
Bmp	Stream for the graphic, stored as an array of charactersi in a <code>OLEVARIANT</code> . Graphic format is Bitmap in the default resolution of the qs-STAT server.

2.4.4.2 GetGraphicExt

The extended version of the `GetGraphic` Command is named `GetGraphicExt`. It has some more functionalities to navigate through individual values and to change the output graphic formats

2.4.4.2.1 IDL syntax

```
[id(0x0000001B)]
HRESULT _stdcall GetGraphicExt(
    [in] long handle,
    [in] long GraphicNr,
    [in] long GraphicSubNr,
    [in] VARIANT PartNr_x,
    [in] VARIANT CharNr_x,
    [in] VARIANT ValueNr_x,
    [in] VARIANT NrOfValues_x,
    [in] VARIANT Group_x,
    [in] VARIANT PartNr_y,
    [in] VARIANT CharNr_y,
    [in] VARIANT ValueNr_y,
    [in] VARIANT NrOfValues_y,
    [in] VARIANT Group_y,
    [in] long Width,
    [in] long Height,
    [in] long NrOfColumns,
    [in] long NrOfRows,
    [in] long ConfigID,
    [in] long OutputFormat,
    [out] VARIANT* Bitmap );
```

2.4.4.2.2 Parameters

Handle	Qs-STAT handle got by the client connect command
GraphicNr	Unique key for a certain desired graphic (See <code>IEnumGraphics</code>)
GraphicSubNr	Subkey, represented specific kinds of the graphic addressed by <code>GraphicNr</code> , usually = 0
PartNr_x, CharNr_x,	Addressing a certain characteristic (primarily)
GroupNr_x	Reserved for future developments
ValueNr_x	Addressing certain individual values (for specific graphic keys only)
..._y	See ..._x for addressing a second characteristic in 2-dimensional graphics (e.g. X-Y Plots)
NrOfValues_...	Reserved for future developments
Width, height	Dimensions of the desired graphic in pixel
NrOfColumns	Reserved for future developments
NrOfRows	Reserved for future developments
ConfigId	Reserved for future developments

OutputFormat	Define the kind of graphic format, that will be generated, see table below
Bitmap	The returned graphic as a stream , stored as an array of characters in a OLEVARIANT. The generated graphic format is defined at the OutputFormat

OutputFormat:

Currently supported are the following graphic formats:

OutputFormat	Graphic format
0	Bitmap (bmp)
1	Jpeg (jpg)
2	Enhanced metafile (EMF)
3	Metafile (WMF)

Remarks:

The parameters to address parts, groups, characteristics, values are defined as VARIANT for future developments to pass lists of characteristics (array of long). Currently they are used as long parameters (see GetGraphic command).

2.4.5 Getting or printing predefined reports

This group of command allows client application to integrate the printable reports of qs-STAT as a graphic into the client user interface. It returns the reports in the same way to the client as the `GetGraphicXXX` methods return the graphics. The available reports can read by the `IEnumReport` interface.

The reports can also be printed to a destination printer by using the group of the `PrintReportxxx` commands, while a list of known printers can be obtained by the `IEnumPrinter` interface.

2.4.5.1 GetReportXXX

2.4.5.1.1 IDL syntax

```
[id(0x00000037)]
HRESULT _stdcall GetReportExt( [in] long      handle,
                              [in] BSTR      PrintReport,
                              [in] long      Part_Nr_x,
                              [in] long      Char_Nr_x,
                              [in] long      Value_Nr_x,
                              [in] long      NrOfValues_x,
                              [in] long      Part_Nr_y,
                              [in] long      Char_Nr_y,
                              [in] long      Value_Nr_y,
                              [in] long      NrOfValues_y,
                              [in] long      page_nr,
                              [in] long      Width,
                              [in] long      Height,
                              [in] long      Config_id,
                              [in] long      OutputFormat,
                              [out] VARIANT* Bitmap );
```

2.4.5.1.2 Parameters

Handle	Qs-STAT handle got by the client connect command
PrintReport	The name of the report to be printed. Names of available reports can be obtained by the <code>IEnumReport</code> interface
PartNr_x, CharNr_x,	Addressing a certain characteristic (primarily)
GroupNr_x	Reserved for future developments
ValueNr_x	Addressing certain individual values (for specific graphic keys only)
..._y	See ..._x for adresssing a second characteristic in 2-dimensional graphics (e.g. X-Y Plots)
NrOfValues_*	Reserved
Width, height	Dimesions of the desired graphic

NrOfColumns	Reserved for future developments
NrOfRows	Reserved for future developments
ConfigId	Reserved for future developments
OutputFormat	Define the kind of graphic format, that will be generated, see table below
Bitmap	The returned graphic as a stream , stored as an array of characters in a OLEVARIANT. The generated graphic format is defined at the OutputFormat

OutputFormat:

Currently supported are the following graphic formats:

OutputFormat	Graphic format
0	Bitmap (bmp)
1	Jpeg (jpg)
2	Enhanced metafile (EMF)
3	Metafile (WMF)
4	Portable Document Format (PDF)

Remarks:

The parameters to adress parts, groups, characteristics, values are defined as VARIANT for future developments to pass lists of characteristics (array of long). Currently they are used as long parameters (see GetGraphic command).

2.4.5.2 GetReportPages

By a call to this method, the client can get the number of available pages for a certain report. This can be used to implement a selection for the page to be displayed.

2.4.5.2.1 IDL syntax

```
[id(0x0000001D)]
HRESULT _stdcall GetReportPages(    [in] long handle,
                                     [in] BSTR PrintReport,
                                     [out] long * NrOfPages );
```

2.4.5.2.2 Parameters

Handle	Qs-STAT handle got by the client connect command
--------	--

PrintReport	The name of the report to be printed. Names of available reports can be obtained by the IEnumReport interface
NrOfPages	Return value, which indicates how many pages are available for a certain report

2.4.5.3 PrintReportXXX

This methods prints a certain report an a certain destination printer. The printer has to be available at the machine, where the qs-STAT server runs. A list of available printers can be obtained by using the IEnumPrinter method.

2.4.5.3.1 IDL syntax

```
[id(0x00000024)]
HRESULT _stdcall PrintReportExt(
    [in] long    handle,
    [in] long    Report_Nr,
    [in] BSTR    Report_Filename,
    [in] long    Printer_ID,
    [in] BSTR    page_nrs,
    [in] long    PrintMode,
    [in] long    ColorMode,
    [in] long    Orientation );
```

2.4.5.3.2 Parameters

Handle	Qs-STAT handle got by the client connect command
Report_nr	Reserved for future developments
Report_filename	Name of the report to be printed -> see IEnumReports
Printer_ID	Key for the destination printer -> see IEnumPrinter
Page_nrs	Pages to be printed formatted in a String (e.g. "1" "1,2")
Print_mode	To define certain ways of generating printouts (see also qs-STAT system settings). Parameters see table below
Color_mode	Switching to colored printout, see table below
Orientation	Orientation of the printout, see table below

Print_Mode:

Value	meaning
31	Printing via intermediate format WMF
32	Printing via intermediate format BMP
33	Direct printing
else	Acc. To qs-STAT system settings (default)

Color_Mode:

value	Meaning
0	Acc. To qs-STAT system settings (default)
1	Color print
2	Black on white
3	White on black
4	Color print / blank background

Orientation:

Not supported

Value	Meaning
0	Default according to report setup

2.4.6 Getting numerical results

By using this method the client can get the available statistical result numerically. Each statistical result has a unique number, that has to be passed by the client. The available results and their keys maybe obtained by the `IEnumStatResults` interface. The results are passed in form of a string and also as a double value, if the result is a number or a floating point value.

The return code is a succeded `HRESULT`, if statistical result and the [out] parameters are valid.

2.4.6.1 GetStatResults

This command is an easy way to obtain statistical (numerical or textual) results from the qs-STAT server for a certain characteristic of the loaded and evaluated data.

2.4.6.1.1 IDL Syntax

```
[id(0x00000003)]
HRESULT _stdcall GetStatResults(    [in] long        handle,
                                   [in] long        StatResultKey,
                                   [in] long        part_Nr,
                                   [in] long        char_Nr,
                                   [in] unsigned char bTransformation,
                                   [out] BSTR*       StatResult_str,
                                   [out] double*     StatResult_dbl );
```

2.4.6.1.2 Parameters

Handle	Qs-STAT handle got by the client connect command
StatResultKey	Unique key for a certain statistical result, keys can be obtained by the <code>IEnumStatResults</code> interface
Part_Nr	Internal number of the certain part of the loaded data
Char_Nr	Internal number of the certain characteristic of the loaded data
BTransformation	Kind of transformation of the data 0: not transformed value 1: transformed value (pls. refer to qs-STAT documentation)
StatResult_str	Numerical result formatted to a string according to the qs-STAT rules
StatResult_dbl	Numerical result in form of a double value

2.4.6.2 GetStatResultExt

To be able to request more detailed information about certain statistical result fields, the command `GetStatResultExt` is provided by the qs-STAT server. This command might return more than one result, according to the presettet parameters (`ResultKey`, `ResultSubKey`, `OutputType`,...). The command returns the values in two way: either as a string value or as a double value, if possible.

This explanation describes the usage of the most significant parameters only, since we cannot cover all properties of all the fields within this document. If a reader is interested in more detailed informations about certain result fields, contact Q-DAS directly and provide the informations, that .you want to get from the server.

2.4.6.2.1 IDL Syntax

```
[id(0x00000020)]
HRESULT _stdcall GetStatResultExt(    [in] long        handle,
                                       [in] long        Part_Nr,
                                       [in] long        Char_Nr,
                                       [in] long        Group_Nr,
                                       [in] long        ResultKey,
                                       [in] long        ResultSubKey,
                                       [in] long        OutputType,
                                       [in] long        TransformationType,
                                       [in] long        reserve1,
                                       [in] long        reserve2,
                                       [out] BSTR *      StatResult_str1,
                                       [out] BSTR *      StatResult_str2,
                                       [out] BSTR *      StatResult_str3,
                                       [out] BSTR *      StatResult_str4,
                                       [out] BSTR *      StatResult_str5,
                                       [out] long *       OutputCount,
                                       [out] double *     StatResult_dbl1,
                                       [out] double *     StatResult_dbl2,
                                       [out] double *     StatResult_dbl3,
                                       [out] double *     StatResult_dbl4,
                                       [out] double *     StatResult_dbl5 );
```

2.4.6.2.2 Parameters

handle	Qs-STAT handle got by the client connect command
Part_Nr, Char_Nr, Group_Nr	Parameters to navigate through the data and access certain characteristics, the char_nr are defined relative to the part_nr parameter.
ResultKey	Unique key for a certain statistical result, keys can be obtained by the <code>IEnumStatResults</code> interface
ResultSubKey	Subkey to access certain "derivated" results for a certain key

OutputType	Bit combined number to obtain certain additional informations about certain fields contents
TransformationType	To distinguish between results for certain kinds of distibutions. See table below
Reserve1, reserve2	Reserved
StatResult_strX	Up to five results formatted as a string, depending on the OutputType parameter
OutputCount	Number of available results, corresponds to StartResult_strX and StatResult_dblX
StatResult_dblX	Up to five results as double values (not at all result-keys available)

OutputType

The parameters are bit coded and the following entries are written in hexadecimal numbers. This table represents the most significant numbers only.

Value	Meaning
\$00000004	Request the content of a field
\$00000040	Request the content of a field including confidence intervall and point estimator
\$00001000	Request the content of a field including confidence intervall without point estimator
\$00200000	Reuquest the content of a filed formatted acc. To specifications of the ASCII-Transferformat (to use in combination with reuquesting the content of a field \$00000040). This is used for certain fields only (e.g. QCC Parameters)

TransformationType

Value	meaning
0	Request not transformed results
1	Request transformed results according to the distribution model
2	Request re-transformed results according to the distribution model

2.5 Navigating in the displayed graphics

The server offers the possibility to client application to navigate inside the created graphics. This is a helpful function for interactions at the user interface of the client application. The server offers two possibilities for the client:

- Decode a cursor position in a certain previously requested graphic
- Mark a certain "data" inside a certain garphic.

These function can be used in a sequence, to obtain the "coordinates" of the cursor (getting part no, characteristic no, value no) firstly and redraw that graphic in combination with marking that data (coordinates), which was obtained by the first step.

2.5.1 Decode the cursor position

This method allows a client to get information about the point in the data (part, characteristic, value) , where the user has placed the e.g. mouse cursor. Therefor the client has to pass the screen coordinates to the server (relative to the origin of a certain graphic) and gets the position inside the data back (part, characteristic, value). This functionality allows interactive navigation inside the data. The client has to give the parameters for the graphic, where the client wants to navigate (identifiers for the graphic, dimensions, data to be displayed).

2.5.1.1 IDL Syntax

```
[id(0x0000003B)]
HRESULT _stdcall GetDataPositionByCoord( [in] long      handle,
                                         [in] long      GraphicNr,
                                         [in] long      GraphicSubNr,
                                         [in] VARIANT    PartNr_x,
                                         [in] VARIANT    CharNr_x,
                                         [in] VARIANT    ValueNr_x,
                                         [in] VARIANT    NrOfValues_x,
                                         [in] VARIANT    GroupNr_x,
                                         [in] VARIANT    PartNr_y,
                                         [in] VARIANT    CharNr_y,
                                         [in] VARIANT    ValueNr_y,
                                         [in] VARIANT    NrOfValues_y,
                                         [in] VARIANT    GroupNr_y,
                                         [in] long      NrOfColumns,
                                         [in] long      NrOfRows,
                                         [in] byte      GetPositionOnly,
                                         [in, out]      long * X,
                                         [in, out]      long * Y,
                                         [in] long      Width,
                                         [in] long      Height,
                                         [in] long      ConfigID,
                                         [in] long      OutputFormat,
```



```

[out] long *      PartNrOut_x,
[out] long *      GroupNrOut_x,
[out] long *      CharNrOut_x,
[out] long *      ValueNrOut_x,
[out] long *      PartNrOut_y,
[out] long *      GroupNrOut_y,
[out] long *      CharNrOut_y,
[out] long *      ValueNrOut_y,
[out] VARIANT *   Bitmap );

```

2.5.1.2 Parameters

The return code is a succeeded HRESULT, if the x,y position is next to a data position (x,y is a valid data position), otherwise the function returns an error.

X, Y	Screen position relative to the origin of the graphic. The server tries to decode this position to data positions for part, char., value. If there is a hit in the data, these positions are changed to the nearest data position inside the graphic (-> [in,out] parameter).
GetPositionOnly	If set to 1, the server doesn't draw the graphic. It's recommended to set this parameter, to get a better performance when the client wants to decode the cursor position only.
PartNrOut_x, CharNrOut_x, ValueNrOut_x	coordinates of a hit data position, only valid, if the return code of the function is a succeeded HRESULT
PartNrOut_y, CharNrOut_y, ValueNrOut_y	coordinates of a hit data position, only valid, if the return code of the function is a succeeded HRESULT, only at x-y plots in use
Bitmap	The requested graphic in a stream in the requested graphic format (OutputFormat). The Bitmap is valid depending on the parameter GetPositionOnly


GetPositionOnly

Using this parameter, the client can decide if he want to

- Draw a certain graphic
- Decode the cursor position into data positions
- Decode the cursor position and draw in one step

The best performance will be reached, if the methods decodes the position only

Value	Meaning
0	Creates the requested graphic (see GetGraphicExt method)
1	Decodes the cursor position (x,y) relative to the origin of a previously created graphic, and returns the "data position (part, char, value). A

	<p>Commands of the qs-STAT Statistical Server</p>	<p>Page 34 / 81</p>
---	---	---------------------

	<p>graphic isn't created and the [out] parameter Bitmap is not valid</p>
<p>2</p>	<p>Combines option 0 and option 1 in one step</p>

2.5.2 Mark a data position

In combination with 2.5.1 this method can be used to request a certain graphic and mark a certain data position inside the graphic (e.g. by drawing a bar in a value plot at a certain value). The marking takes place in a certain range of the data, that has to be defined by the client application.

How the given data position is marked is a system setting of the qs-STAT server. For detailed description, see 2.4.4.2. The following describes the extensions of this method only.

2.5.2.1 IDL Syntax

```
[id(0x0000003C)]
HRESULT _stdcall GetGraphicAndMark(
    [in] long    handle,
    [in] long    GraphicNr,
    [in] long    GraphicSubNr,
    [in] VARIANT PartNr_x,
    [in] VARIANT GroupNr_x,
    [in] VARIANT CharNr_x,
    [in] VARIANT ValueNr_x,
    [in] VARIANT NrOfValues_x,
    [in] VARIANT PartNr_y,
    [in] VARIANT GroupNr_y,
    [in] VARIANT CharNr_y,
    [in] VARIANT ValueNr_y,
    [in] VARIANT NrOfValues_y,
    [in] long    PartNr_MarkStart,
    [in] long    GroupNr_MarkStart,
    [in] long    CharNr_MarkStart,
    [in] long    ValueNr_MarkStart,
    [in] long    PartNr_MarkEnd,
    [in] long    GroupNr_MarkEnd,
    [in] long    CharNr_MarkEnd,
    [in] long    ValueNr_MarkEnd,
    [in] long    NrOfColumns,
    [in] long    NrOfRows,
    [in] long    Width,
    [in] long    Height,
    [in] long    ConfigID,
    [in] long    OutputFormat,
    [out] VARIANT * Bitmap );
```

2.5.2.2 Parameters

The return code is a succeeded HRESULT, if the the requested graphic was created I the [out] parameter Bitmap, otherwise the function returns an error (e.g. if an invalid graphic key was used). The parameters are similar to the GetGraphicExt method. This methods returns a graphic in the requested format (OutputFormat).

GraphicNr, GraphicSubNr	Type of graphic, that shall be generated, see IEnumGraphic
PartNr_MarkStart, GroupNr_MarkStart, CharNr_MarkStart, ValueNr_MarkStart	Data position, where the reange of the marked data starts. This parameter can be obtained by a call to GetDataPositionByCoordinates
PartNr_MarkEnd, GroupNr_MarkEnd, CharNr_MarkEnd, ValueNr_MarkEnd	Data position, where the reange of the marked data ends. This parameter can be obtained by a call to GetDataPositionByCoordinates
OutputFormat	Graphic format, in which the graphic will be generated (see GetGraphicExt)
Bitmap	The requested graphic in a stream in the requested graphic format (OutputFormat). The Bitmap is valid depending on the parameter GetPositionOnly

2.6 Special functions

In addition, here comes a list of functionalities of the server's interface, which cannot be categorized.

2.6.1 TakeoverQCC

This method gives control to an external system for using calculated or stored QCCs as that QCC, which is used for alarm detection in an online system for a certain characteristic. For more details about the QCC usage inside Q-DAS products, please refer also to the qs-STAT / procella documentation.

2.6.1.1 IDL-Syntax

```
[id(0x00000045)]
HRESULT _stdcall TakeoverOnlineQCC( [in] long      handle,
                                     [in] VARIANT PartNr,
                                     [in] VARIANT CharNr,
                                     [in] long      QCCSource )
```

2.6.1.2 Parameters

The return code is a succeded HRESULT, if the the requested graphic was created I the [out] parameter Bitmap, otherwise the function returns an error (e.g. if an invalid graphic key was used). The parameters are similar to the GetGraphicExt method. This methods returns a graphic in the requested format (OutputFormat).

handle	The current connection
PartNr, CharNr	The characteristic or the characteristics, for which the operation should be performed
QCCSource	A identifier for the QCC, that should be taken over as an Online QCC/alarm QCC: Valid values are: 1: analysis QCC (calculated based on process capability calculation, "Online Analysis Server" only) 2: stored QCC (given form the data set by K-Fields K8xxx) 3: SPC-QCC (calculated based on process capability calculation, "Online Analysis Server" only)

2.6.1.3 Remarks

This method is in use at module 26 only ("Online Server" or "Online Analysis Server"). Setting parameter QCCSource to 3 (SPC QCC) or 1 (analysis QCC) is recommended only immediately after a successful call to EvaluateCharXXX methods for the effected characteristics (including the corresponding Event).

3 Accessing the Q-DAS Database

The qs-STAT COM interface offers a lot of possibilities of querying the Q-DAS database. All functions needed to do so can be found in the interface

```
IDataBase
[
    uuid(11853321-12CB-11D5-B22D-00105AD88C31),
    version(1.0),
    dual,
    oleautomation
]
```

The central object of database access is the query. To receive data, a query has to be created or loaded first. The query can be modified then by adding different kinds of filters to specify the desired data. Finally it is executed, which means that it is loaded to qs-STAT, ready for evaluation, or information about parts, characteristics or measurement values can be received.

The interface supports the following functions:

3.1.1 GetFirstQueryName

This function offers, together with `GetNextQueryName`, the possibility to get a collection of names of database queries which once have been stored.

3.1.1.1 IDL Syntax

```
[id(0x00000001)]
HRESULT _stdcall GetFirstQueryName(    [in]    long handle,
                                       [out]   BSTR * QueryName );
```

3.1.1.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryName`
Name of the database query returned from the qs-STAT server

3.1.2 GetNextQueryName

Once having received a query name with GetFirstQueryName, further query names can be received using this function as long as the function returns 0.

3.1.2.1 IDL Syntax

```
[id(0x00000002)]
HRESULT _stdcall GetNextQueryName(           [in]      long handle,
                                              [out]     BSTR * QueryName );
```

3.1.2.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryName`
Name of the database query returned from the qs-STAT server

3.1.3 LoadQuery

This function activates a stored query so that it can be executed in a further step. It does not yet load data for evaluation.

3.1.3.1 IDL Syntax

```
[id(0x00000003)]
HRESULT _stdcall LoadQuery(                  [in]      long handle,
                                              [in]      BSTR QueryName,
                                              [out]     long * QueryHandle );
```

3.1.3.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryName`
Name of the database query previously received
3. `QueryHandle`
Handle of the database query created on the qs-STAT server

3.1.4 CreateQuery

This function creates a new query which can be executed in a further step. It can receive querying information using the commands `AddFilterToQuery`, `AddSortToQuery` or `AddPartCharacteristicListToQuery`.

3.1.4.1 IDL Syntax

```
[id(0x00000004)]
HRESULT _stdcall CreateQuery(
    [in]      long handle,
    [out]     long * QueryHandle );
```

3.1.4.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query created on the qs-STAT server

3.1.5 AddFilterToQuery

This function adds an existing filter a query. The filter might previously have been loaded using the `LoadFilter` command or created using one of the `CreateFilter` commands. To a query there can be added up to three filters, each according to the three sensible values of `QueryLevel`: `PART_LEVEL_C` (= 0), `CHARACTERISTIC_LEVEL_C` (= 1), `VALUE_LEVEL_C` (= 2). The parameter `QueryLevel` describes how the filter is used. Example: A filter created to select data by a range of measurement date can be used to get measurement values belonging to that certain range. In this case the parameter `QueryLevel` is set to `VALUE_LEVEL_C`. But it could also be used to select all parts which have measurement values in that range. For that purpose `QueryLevel` would have to be set to `PART_LEVEL_C`. The parameters `Part_Key` and `Char_Key` can specify that a filter is only used on a specific part or on a specific combination of part and characteristic. If the filter is to be used without part or characteristic specification, these parameters have to be 0.

3.1.5.1 IDL Syntax

```
[id(0x00000005)]
HRESULT _stdcall AddFilterToQuery(
    [in]      long handle,
    [in]      long QueryHandle,
    [in]      long FilterHandle,
    [in]      unsigned char QueryLevel,
```



```
[in]      long Part_Key,
[in]      long Char_Key );
```

3.1.5.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `FilterHandle`
Handle of a filter received from the LoadFilter command or one of the CreateFilter commands
4. `QueryLevel`
Level on which the added filter is to operate.
Sensible values: PART_LEVEL_C (= 0), CHARACTERISTIC_LEVEL_C (= 1),
VALUE_LEVEL_C (= 2)
5. `Part_Key`
Database key of the part the filter is to be used on, or 0 if the filter is to be used on all parts
6. `Char_Key`
Database key of the characteristic the filter is to be used on, or 0 if the filter is to be used on all characteristics

3.1.6 AddSortToQuery

This function allows to sort data before they are received from the query. Because it needs some information about how to sort the data, there is the parameter SortKey, which contains a K-Field number to sort by. The parameter Direction tells the query whether to sort in ascending (Direction = SORT_ASCENDING_C = 0) or descending order (Direction = SORT_DESCENDING_C = 1).

3.1.6.1 IDL Syntax

```
[id(0x00000006)]
HRESULT _stdcall AddSortToQuery(
    [in]      long handle,
    [in]      long QueryHandle,
    [in]      long SortKey,
    [in]      unsigned char Direction);
```

3.1.6.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`

Handle of the database query received in one of the previous steps

3. SortKey

Qs-STAT key of the field to sort by, e. g. 4 if you want to sort by date/time (K0004)

4. Direction

Sorting direction

Sensible values: SORT_ASCENDING_C (= 0), SORT_DESCENDING_C (= 1)

3.1.7 AddFunctionToQuery

This function is not yet implemented.

3.1.7.1 IDL Syntax

```
[id(0x00000007)]
HRESULT _stdcall AddFunctionToQuery(
    [in] long handle,
    [in] long QueryHandle,
    [in] long FunctionHandle,
    [in] unsigned char QueryLevel,
    [in] long Part_Key,
    [in] long Char_Key );
```

3.1.8 AddAutoSelectToQuery

This function has not been implemented. Instead, special filters can be created to perform an “automatic selection”.

3.1.8.1 IDL Syntax

```
[id(0x00000008)]
HRESULT _stdcall AddAutoSelectToQuery(
    [in] long handle,
    [in] long QueryHandle,
    [in] long AutoSelectHandle,
    [in] unsigned char QueryLevel,
    [in] long Part_Key,
    [in] long Char_Key );
```

3.1.9 SetQueryProperty

This function offers the possibility to set certain properties before the query is executed, e.g. several selected parts can be joined together so that they can be evaluated as one part.

3.1.9.1 IDL Syntax

```
[id(0x00000011)]
HRESULT _stdcall SetQueryProperty(           [in]      long handle,
                                              [in]      long QueryHandle,
                                              [in]      long PropertyKey,
                                              [in]      BSTR PropertyValue );
```

3.1.9.2 Parameters

1. **handle**
Qs-STAT handle received by the client connect command
2. **QueryHandle**
Handle of the database query received in one of the previous steps
3. **PropertyKey**
An integer value indicating one of the properties listed below
4. **PropertyValue**
The new value of the property as a string. Possible values are listed below.

Value of PropertyKey	Property	Possible Values of PropertyValue, Explanation
5001	LogFileName	Path and file name of a log file. The log file yields information on database queries performed during execution of the "ExecuteQuery" function. An empty string resets the property so that no log file is written.
5010	PartJoin	"1" or "TRUE" to set, "0" or "FALSE" to reset property. When the property is set, the parts selected to query are loaded as one part, and characteristics of those parts are also joined together if they match in Fields K2001 and K2002.
5011	CharJoin	"1" or "TRUE" to set, "0" or "FALSE" to reset property. When the property is set, the characteristics selected to query are loaded as one characteristic, containing all measurement values. CharJoin includes PartJoin automatically.
5012	PartKeyToJoinTo	The database key of the part that will be considered as "master part" when using the PartJoin option. The field data of the part defined by this key will be loaded to the "joined" part. If omitted, field data will be taken from the first part found.
5015	Diff_2Parts	"1" or "TRUE" to set, "0" or "FALSE" to reset property. This option is only sensible when two parts are loaded which have the same set of characteristics. The two parts are joined together similarly to "PartJoin", but then differences are calculated between corresponding measurement values.
5030	AutoSelDuplValues	"1" or "TRUE" to set, "0" or "FALSE" to reset property. When an automatic query is performed, which separates the measurement values into sub-characteristics according to additional data such as date/time, this option provides copies of

		all the measurement values to the master (original) characteristics.
5050	FittingBatchNos	"1" or "TRUE" to set, "0" or "FALSE" to reset property. When the property is set, measurement values are sorted by their batch numbers (when used as serial numbers). Additionally, if certain serial numbers are missing in some of the characteristics, they are filled up with "invalid" values.
5080	UseFilterOnPartList	"1" or "TRUE" to set, "0" or "FALSE" to reset property. If a part list is given and a filter is set on part level, this option shrinks the list of parts to those which fit into the filter condition. Otherwise the part list overrides the filter.
5090	LoadWithoutValues	"1" or "TRUE" to set, "0" or "FALSE" to reset property. If this option is set, only part and characteristic data are loaded

3.1.10 SaveQuery

This function is used to store a query under a specified name.

3.1.10.1 IDL Syntax

```
[id(0x00000009)]
HRESULT _stdcall SaveQuery(
                                [in]      long handle,
                                [in]      BSTR QueryName,
                                [in]      long QueryHandle );
```

3.1.10.2 Parameters

5. `handle`
Qs-STAT handle received by the client connect command
6. `QueryName`
The name of the query is to be stored under
7. `QueryHandle`
Handle of the database query received in one of the previous steps

3.1.11 ExecuteQuery

This function loads the data specified in previous steps into qs-STAT's memory structures, ready for evaluation. A list of parts and characteristics can be given additionally.

3.1.11.1 IDL Syntax

```
[id(0x0000000A)]
HRESULT _stdcall ExecuteQuery(
    [in]      long handle,
    [in]      long QueryHandle,
    [in]      VARIANT Part_Char_List );
```

3.1.11.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `Part_Char_List`
OLE variant containing a list of parts or parts and characteristics to be loaded.
It can be a null variant, i.e. a variant initialized with `vt = VT_NULL` in C++ or simply the NULL variant in Delphi; in this case the query is executed completely.
It can also be a one-dimensional array containing part database keys (type: long); in this case the query is reduced to the specified parts. The part keys can be received from the functions `GetFirstPartQuery/GetNextPartQuery`.
Finally, it can also be a two-dimensional array containing part database keys in fields `[i, 0]` and corresponding characteristic database keys in fields `[i, j]` ($j > 0$). Because the different parts in the list may require different numbers of characteristics, some field values can be zero, which means they do not carry information about a characteristic. If the value of field `[i, 1]` is set to -1, the part defined in field `[i, 0]` is to be loaded with all its characteristics, as it would be if a one-dimensional array was used.

3.1.12 GetFirstPartQuery

This function returns specific information about the first part of a query. On Success it returns 0.

3.1.12.1 IDL Syntax

```
[id(0x0000000B)]
HRESULT _stdcall GetFirstPartQuery(
    [in]      long handle,
    [in]      long QueryHandle,
    [in]      VARIANT KFieldList,
    [out]     long * Part_Key,
    [out]     VARIANT * KResultList);
```

3.1.12.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `KFieldList`
OLE variant containing a one-dimensional array of qs-STAT K-fields (type: long) which are to be returned
4. `Part_Key`
Returns the part database key
5. `KResultList`
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to `KFieldList`.

3.1.13 GetNextPartQuery

After having called `GetFirstPartQuery`, this function returns specific information about the following parts of a query, provided that `GetFirstPartQuery` returned 0. `GetNextPartQuery` can be called repeatedly to get a list of part information until it returns nonzero.

3.1.13.1 IDL Syntax

```
[id(0x0000000C)]
HRESULT _stdcall GetNextPartQuery(
    [in]         long handle,
    [in]         long QueryHandle,
    [out]        long * Part_Key,
    [out]        VARIANT * KResultList);
```

3.1.13.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `Part_Key`
Returns the part database key
4. `KResultList`
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to `KFieldList` defined in `GetFirstPartQuery`.

3.1.14 SkipPartQuery

This function works like GetNextPartQuery, but before returning a result it skips a specified number of part datasets.

3.1.14.1 IDL Syntax

```
[id(0x0000000D)]
HRESULT _stdcall SkipPartQuery(
                                [in]      long handle,
                                [in]      long QueryHandle,
                                [in]      long num,
                                [out]     long * Part_Key,
                                [out]     VARIANT * KResultList);
```

3.1.14.2 Parameters

1. **handle**
Qs-STAT handle received by the client connect command
2. **QueryHandle**
Handle of the database query received in one of the previous steps
3. **num**
The number of datasets to be skipped
4. **Part_Key**
Returns the part database key
5. **KResultList**
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to KfieldList defined in GetFirstPartQuery.

3.1.15 GetFirstCharQuery

This function returns specific information about the first characteristic of a specified part belonging to a query. On Success it returns 0.

3.1.15.1 IDL Syntax

```
[id(0x0000000E)]
HRESULT _stdcall GetFirstCharQuery(
                                [in]      long handle,
                                [in]      long QueryHandle,
                                [in]      VARIANT KFieldList,
                                [in]      long Part_Key,
                                [out]     long * Char_Key,
                                [out]     VARIANT * KResultList);
```

3.1.15.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `KFieldList`
OLE variant containing a one-dimensional array of qs-STAT K-fields (type: long) which are to be returned
4. `Part_Key`
The part database key for which characteristics are to be found
5. `Char_Key`
Returns the characteristic database key
6. `KResultList`
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to `KFieldList`.

3.1.16 GetNextCharQuery

After having called `GetFirstCharQuery`, this function returns specific information about the following characteristics of a query, provided that `GetFirstCharQuery` returned 0. `GetNextCharQuery` can be called repeatedly to get a list of characteristic information until it returns nonzero.

3.1.16.1 IDL Syntax

```
[id(0x0000000F)]
HRESULT _stdcall GetNextCharQuery(
    [in]         long handle,
    [in]         long QueryHandle,
    [out]        long * Char_Key,
    [out]        VARIANT * KResultList);
```

3.1.16.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `Char_Key`

Returns the characteristic database key

4. **KResultList**

Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to KfieldList defined in GetFirstCharQuery.

3.1.17 SkipCharQuery

This function works like GetNextCharQuery, but before returning a result it skips a specified number of characteristic datasets.

3.1.17.1 IDL Syntax

```
[id(0x00000010)]
HRESULT _stdcall SkipCharQuery(
                                [in]      long handle,
                                [in]      long QueryHandle,
                                [in]      long num,
                                [out]     long * Char_Key,
                                [out]     VARIANT * KResultList);
```

3.1.17.2 Parameters

1. **handle**
Qs-STAT handle received by the client connect command
2. **QueryHandle**
Handle of the database query received in one of the previous steps
3. **num**
The number of datasets to be skipped
4. **Char_Key**
Returns the characteristic database key
5. **KResultList**
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to KfieldList defined in GetFirstCharQuery.

3.1.18 GetFirstValueQuery

This function returns specific information about the first measurement value of a specified part and characteristic belonging to a query. On Success it returns 0.

3.1.18.1 IDL Syntax

```
[id(0x00000020)]
HRESULT _stdcall GetFirstValueQuery(
                                [in]      long handle,
```

```

[in]      long QueryHandle,
[in]      VARIANT KFieldList,
[in]      long Part_Key,
[in]      long * Char_Key,
[out]     long * Value_Key,
[out]     VARIANT * KResultList);

```

3.1.18.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `KFieldList`
OLE variant containing a one-dimensional array of qs-STAT K-fields (type: long) which are to be returned
4. `Part_Key`
The part database key for which measurement values are to be found
5. `Char_Key`
The characteristic database key for which measurement values are to be found
6. `Value_Key`
Returns the measurement value database key
7. `KResultList`
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to KfieldList.

3.1.19 GetNextValueQuery

After having called `GetFirstValueQuery`, this function returns specific information about the following measurement values of a query, provided that `GetFirstValueQuery` returned 0. `GetNextValueQuery` can be called repeatedly to get a list of measurement value information until it returns nonzero.

3.1.19.1 IDL Syntax

```

[id(0x00000030)]
HRESULT _stdcall GetNextValueQuery(
[in]      long handle,
[in]      long QueryHandle,
[out]     long * Value_Key,
[out]     VARIANT * KResultList);

```

3.1.19.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `Value_Key`
Returns the database key of the measurement value
4. `KResultList`
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to `KfieldList` defined in `GetFirstValueQuery`.

3.1.20 SkipValueQuery

This function works like `GetNextValueQuery`, but before returning a result it skips a specified number of measurement value datasets.

3.1.20.1 IDL Syntax

```
[id(0x00000040)]
HRESULT _stdcall SkipValueQuery(
    [in]      long handle,
    [in]      long QueryHandle,
    [in]      long num,
    [out]     long * Value_Key,
    [out]     VARIANT * KResultList);
```

3.1.20.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps
3. `num`
The number of datasets to be skipped
4. `Value_Key`
Returns the database key of the measurement value
5. `KResultList`
Returns an OLE variant containing a one-dimensional array of result strings (BSTR) corresponding to `KfieldList` defined in `GetFirstValueQuery`.

3.1.21 FreeQuery

This function releases memory space used by a query when it is no longer used.

3.1.21.1 IDL Syntax

```
[id(0x00000050)]
HRESULT _stdcall FreeQuery(
                                [in]      long handle,
                                [in]      long QueryHandle );
```

3.1.21.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `QueryHandle`
Handle of the database query received in one of the previous steps

3.1.22 GetFirstFilterName

Provided that filters have been created and stored before, this function can be used to start receiving a list of them. If its result is zero, it returns the first filter name in its output parameter. Further filter names can be received using the `GetNextFilterName` function.

3.1.22.1 IDL Syntax

```
[id(0x00000060)]
HRESULT _stdcall GetFirstFilterName(
                                [in]      long handle,
                                [out]     BSTR * FilterName );
```

3.1.22.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `FilterName`
Name of the first stored filter

3.1.23 GetNextFilterName

Provided that previous function calls of `GetFirstFilterName` or `GetNextFilterName` returned zero, this function returns the next filter name in its output parameter. The call of `GetNextFilterName` can be repeated until its result is nonzero.

3.1.23.1 IDL Syntax

```
[id(0x00000070)]
HRESULT _stdcall GetNextFilterName(           [in]      long handle,
                                              [out]     BSTR * FilterName );
```

3.1.23.2 Parameters

1. handle
Qs-STAT handle received by the client connect command
2. FilterName
Next name of a stored filter

3.1.24 LoadFilter

This function activates a stored filter object. The handle which is returned, can be used for querying in a further step.

3.1.24.1 IDL Syntax

```
[id(0x00000080)]
HRESULT _stdcall LoadFilter(                 [in]      long handle,
                                              [in]      BSTR FilterName,
                                              [out]     long * FilterHandle );
```

3.1.24.2 Parameters

1. handle
Qs-STAT handle received by the client connect command
2. FilterName
Name of the filter which is to be loaded
3. FilterHandle
Returns the handle of the filter object

3.1.25 CreateFilterFromFilters

This function allows to create a filter as a combination of other filters which have been created or loaded before. The filters are combined using a specified Boolean operator.

3.1.25.1 IDL Syntax

```
[id(0x00000090)]
HRESULT _stdcall CreateFilterFromFilters( [in]      long handle,
                                         [in]      unsigned char Operator,
                                         [in]      VARIANT SourceFilters,
                                         [out]     long * FilterHandle );
```

3.1.25.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `Operator`
Contains a constant representing a Boolean operator:
0 = AND; 1 = OR; 2 = NOT; 3 = NAND; 4 = NOR
3. `SourceFilters`
A one-dimensional array (OLE variant) containing the input list of filter handles
4. `FilterHandle`
Returns the handle of the filter object

3.1.26 CreateFilter

This function allows to create a filter using a description of a condition.

3.1.26.1 IDL Syntax

```
[id(0x000000A0)]
HRESULT _stdcall CreateFilter(
                                         [in]      long handle,
                                         [in]      unsigned char k_r_key,
                                         [in]      long ausgabepunkt,
                                         [in]      BSTR Condition,
                                         [in]      unsigned char CompOp,
                                         [out]     long * FilterHandle );
```

3.1.26.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `k_r_key`
Has to be set to 1 to use Q-DAS K-fields to select
3. `ausgabepunkt`

The K-field number to select by

4. **Condition**

Contains the string to search for. It can also contain a number (decimally converted to a String) or a date (format: YYYY/MM/DD HH:MI:SS).

5. **CompOp**

A constant representing the comparing operator:

0 = equal; 1 = LIKE (comparison by substring); 2 = greater than; 3 = smaller than; 4 = greater than or equal; 5 = smaller than or equal; 128 = get the first n values; 129 = get the last n values

5. **FilterHandle**

Returns the handle of the filter object

3.1.26.3 Examples ?

3.1.27 CreateFilterFromSQL

This function allows to create a filter using an SQL statement as a condition. The SQL statement has to use table names and column names belonging to the Q-DAS database.

3.1.27.1 IDL Syntax

```
[id(0x000000B0)]
HRESULT _stdcall CreateFilterFromSQL(           [in]      long handle,
                                                [in]      BSTR FilterString,
                                                [out]     long * FilterHandle );
```

3.1.27.2 Parameters

1. **handle**

Qs-STAT handle received by the client connect command

2. **FilterString**

Contains the part of an SQL statement which defines the condition (the part of a SELECT statement which starts after the WHERE keyword, without any further declarations such as ORDER BY, GROUP BY, HAVING etc.)

3. **FilterHandle**

Returns the handle of the filter object

3.1.28 SaveFilter

This function is used to store a filter under a specified name.

3.1.28.1 IDL Syntax

```
[id(0x000000C0)]
HRESULT _stdcall SaveFilter(
                                [in]      long handle,
                                [in]      BSTR FilterString,
                                [in]      long FilterHandle );
```

3.1.28.2 Parameters

1. handle
Qs-STAT handle received by the client connect command
2. FilterString
The name the filter is to be stored under
3. FilterHandle
Handle of the filter received in one of the previous steps

3.1.29 FreeFilter

This function releases memory space used by a filter when it is no longer used.

3.1.29.1 IDL Syntax

```
[id(0x000000D0)]
HRESULT _stdcall FreeFilter(
                                [in]      long handle,
                                [in]      long FilterHandle );
```

3.1.29.2 Parameters

1. handle
Qs-STAT handle received by the client connect command
2. FilterHandle
Handle of the filter received in one of the previous steps

3.1.30 CreateDirectSQL

This function creates an object in qs-STAT which is able to perform SQL statements in the database. The ExecuteDirectSQL command will receive the SQL statement itself and execute it in the database.

3.1.30.1 IDL Syntax


```
[id(0x000000E0)]
HRESULT _stdcall CreateDirectSQL(           [in]      long handle,
                                             [out]     long * SQLHandle );
```

3.1.30.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `SQLHandle`
Handle of the new SQL object

3.1.31 ExecuteDirectSQL

After a DirectSQL object has been created, it can receive its command and execute it with this function. If the SQL command is a SELECT statement, the function will return a list of field types of the columns which are affected. In this case the rows returned by the Query can be received using the `GetFirstDirectSQLRow` and `GetNextDirectSQLRow` commands.

3.1.31.1 IDL Syntax

```
[id(0x00000021)]
HRESULT _stdcall ExecuteDirectSQL(           [in]      long handle,
                                             [in]      long SQLHandle,
                                             [in]      BSTR SQLString,
                                             [out]     VARIANT * FieldList );
```

3.1.31.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `SQLHandle`
Handle of the SQL object created with `CreateDirectSQL`
3. `SQLString`
A String containing the SQL statement to be executed
4. `FieldList`
An OLE variant containing a one-dimensional array of long integer values which represent the column field types if the command is a SELECT statement, otherwise it returns NULL.

(ftUnknown, ftString, ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftBytes, ftVarBytes, ftAutoInc, ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor, ftFixedChar, ftWideString, ftLargeint, ftADT, ftArray, ftReference, ftDataSet, ftOraBlob, ftOraClob, ftVariant, ftInterface, ftIDispatch, ftGuid)

3.1.32 GetFirstDirectSQLRow

After a select statement has been executed, this command can receive its first resulting row as an array of strings. If it returns zero, further rows can be received using the GetNextDirectSQLRow command.

3.1.32.1 IDL Syntax

```
[id(0x000000F0)]
HRESULT _stdcall GetFirstDirectSQLRow(      [in]      long handle,
                                             [in]      long SQLHandle,
                                             [out]     VARIANT * ResultList);
```

3.1.32.2 Parameters

1. **handle**
Qs-STAT handle received by the client connect command
2. **SQLHandle**
Handle of the SQL object created with CreateDirectSQL
3. **ResultList**
An OLE variant containing an array of string values (BSTR) which represent the first returned row. The array has to match the FieldList parameter returned by ExecuteDirectSQL. Using the field types in FieldList, the strings in ResultList can be re-transformed to their original data type.

3.1.33 GetNextDirectSQLRow

After a GetfirstDirectSQLRow command succeeded (returned zero), further rows can be received with the GetNextDirectSQLRow command as long as it returns zero.

3.1.33.1 IDL Syntax

```
[id(0x00000100)]
HRESULT _stdcall GetNextDirectSQLRow(      [in]      long handle,
                                           [in]      long SQLHandle,
                                           [out]     VARIANT * ResultList);
```

3.1.33.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `SQLHandle`
Handle of the SQL object created with `CreateDirectSQL`
3. `ResultList`
An OLE variant containing an array of string values (BSTR) which represent the returned row. The array has to match the `FieldList` parameter returned by `ExecuteDirectSQL`. Using the field types in `FieldList`, the strings in `ResultList` can be re-transformed to their original data type.

3.1.34 FreeDirectSQL

When the `DirectSQL` object is not needed any longer, the memory it occupies should be released with this command.

3.1.34.1 IDL Syntax

```
[id(0x00000200)]
HRESULT _stdcall FreeDirectSQL(            [in]      long handle,
                                           [in]      long SQLHandle);
```

3.1.34.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `SQLHandle`
Handle of the SQL object created with `CreateDirectSQL`

3.1.35 RecentSerNo_First

For a certain part the last serial number which can be found in the database (ordered by measurement date/time) can be received.

3.1.35.1 IDL Syntax

```
[id(0x0000001A)]
HRESULT _stdcall RecentSerNo_First(           [in]      long handle,
                                              [in]      long PartKey,
                                              [out]     BSTR * SerialNo );
```

3.1.35.2 Parameters

1. handle
Qs-STAT handle received by the client connect command
2. PartKey
The database key of the concerned part
3. SerialNo
Returns the desired serial number

3.1.36 RecentSerNo_Next

If RecentSerNo_First returned zero, further serial numbers can be received using this function as long as it returns zero. Attention: As this and the preceding function yield the most recent serial numbers, RecentSerNo_Next always gives the serial number **preceding** in date/time.

3.1.36.1 IDL Syntax

```
[id(0x0000001B)]
HRESULT _stdcall RecentSerNo_Next(           [in]      long handle,
                                              [out]     BSTR * SerialNo );
```

3.1.36.2 Parameters

1. handle
Qs-STAT handle received by the client connect command
2. SerialNo
Returns the following (i. e. preceding in time) serial number

3.1.37 Get_n_Buildphases

This and the two following functions offer the possibility to get a special kind of data evaluation and result display. It requires that a product is produced in a row of buildphases from development to grown-up serial production. The so-called Benchmark evaluation shows the development of the capability indices of a large amount of

characteristics throughout the buildphases. The Benchmark evaluation needs evaluation results stored in the database with the SaveResults client program.

This function returns a list of buildphases available for certain parts.

3.1.37.1 IDL Syntax

```
[id(0x0000001C)]
HRESULT _stdcall Get_n_Buildphases(
    [in]      long handle,
    [in]      VARIANT PartKeyList,
    [in]      long Number,
    [out]     VARIANT * BuildPhaseList);
```

3.1.37.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `PartKeyList`
An OLE variant containing a one-dimensional array of integers representing the database part keys buildphases are to be found for
3. `Number`
The maximum number of buildphases to be returned. If the number of buildphases available exceeds the parameter Number, the most recent buildphases are taken.
4. `BuildPhaseList`
An OLE variant containing an array of strings (BSTR) which represent the buildphases

3.1.38 GetBuildphases

This function returns a list of buildphases beginning with a first and ending with a last phase.

3.1.38.1 IDL Syntax

```
[id(0x0000001D)]
HRESULT _stdcall GetBuildphases(
    [in]      long handle,
    [in]      BSTR FirstPhase,
    [in]      BSTR LastPhase,
    [out]     VARIANT * BuildPhaseList );
```

3.1.38.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `FirstPhase`
The buildphase intended to be the first in the list
3. `LastPhase`
The buildphase intended to be the last in the list
4. `BuildPhaseList`
An OLE variant containing an array of strings (BSTR) which represent the buildphases

3.1.39 Benchmark

This function loads specified data and performs a Benchmark evaluation. After that, the Benchmark graphic can be received using the graphic key 7250.

3.1.39.1 IDL Syntax

```
[id(0x0000001E)]
HRESULT _stdcall Benchmark(
    [in]          long handle,
    [in, out]    VARIANT * PartKeyList,
    [in]          VARIANT BuildPhaseList,
    [in] VARIANT OutputPointList);
```

3.1.39.2 Parameters

1. `handle`
Qs-STAT handle received by the client connect command
2. `PartKeyList`
An OLE variant containing an array of integer values representing the database part keys for which the Benchmark evaluation is to be done. It returns a list of parts for which the evaluation has been done (i. e. the same list, but without those parts which could not be evaluated).
3. `BuildPhaseList`
An OLE variant containing an array of strings (BSTR) which represent the buildphases for which the Benchmark is to be done.
4. `OutputPointList`
An OLE variant containing an array of integer values which represent the output points for which the Benchmark is to be done.

3.1.40 OpeningLine2

This function loads data of two parts which are to fit together so that an opening line results. After that, the Benchmark graphic can be received using the graphic key 7250.

3.1.40.1 IDL syntax

```
[id(0x0000001F)]
HRESULT _stdcall OpeningLine2(
    [in]      long handle,
    [in]      BSTR SerialNoA,
    [in]      BSTR SerialNoB,
    [in]      long PartKeyA,
    [in]      long PartKeyB,
    [in]      VARIANT CharPairList );
```

3.1.40.2 Parameters

1. **handle**
Qs-STAT handle received by the client connect command
2. **PartKeyList**
An OLE variant containing an array of integer values representing the database part keys for which the Benchmark evaluation is to be done. It returns a list of parts for which the evaluation has been done (i. e. the same list, but without those parts which could not be evaluated).
3. **BuildPhaseList**
An OLE variant containing an array of strings (BSTR) which represent the buildphases for which the Benchmark is to be done.
4. **OutputPointList**
An OLE variant containing an array of integer values which represent the output points for which the Benchmark is to be done.

3.2 Events

The interface for getting events from the Q-DAS server is named:

- IQsEvents
(uuid(409C3465-74EA-4AF9-82F6-7C33AE0C4912))

The server interacts with the client by using the connection point technology, so the client has to implement the IQsEvents interface. Currently this interface has 4 major methods

- OnQSSTATInitialized
Server informs the client, when he is ready to go.
- OnGetAliveState
The server asks the client, if he is still alive
- OnDataAvailXXX
For synchronization, when sending the data to the server or when loading data from external resources like files or databases.
- OnEvaluationDoneXXX
For synchronisation, when data is evaluated concurrently.

Please note that the interface defines also OnxxxxExt Events to allow a client application to distinguish between the events, if several connections are established at run time. These OnxxxxExt Events pass the connection handle for the connection, where the event happened, back to the client application in addition to the parameters described below.

In case of multiple connections to the server, the OnXXXXExt Events pass the connection handle back to the client. This is the connection handle (data instance), where the Event belongs to. So a client application is able to distinguish between different connections.

3.2.1 Alive State

The client has to implement this method, so that the server can check, if the client application is still running correctly. Otherwise the server may release the internal structures for the client and the client has to call the connection again. To indicate to the server, that the client is running, the client has to set the value for the parameter state to 1.

3.2.1.1 IDL Syntax

```
[id(0x00000001)]
HRESULT _stdcall OnGetAliveState( [out] long* state);
```

3.2.1.2 Parameters

1. state

The client has to set this parameter to 1.

3.2.2 Server Initialization

The client will be informed, when the server is ready to work. This is of interest directly after creating the COM object inside of the Client application. The server needs some time to initialize its data structures and load the configuration. After this work is done, the method `OnQsstatInitialized` is called. The client application has to wait for this signal, before it sends data to the server.

3.2.2.1 IDL Syntax

```
[id(0x00000003)]
HRESULT _stdcall OnQSSTATInitialized( void );
```

3.2.2.2 Parameters

none
.

3.2.3 Data loading Synchronization

This even indicates the client application, that the server has processed a packet of data, that was sent to him by using the `OpenFile`, `TransmitFile`, `DBOpen` and `TransmitPartFile` method. The server is now ready to get the next packet of data, that will be added to the already loaded data or to evaluate the loaded data, if the loading is finished.

3.2.3.1 IDL Syntax

```
[id(0x00000002)]
HRESULT _stdcall OnDataAvail( [in] long NumOfChars );
[id(0x00000009)]
HRESULT OnDataAvailExt([in] long handle, [in] long NumOfChars );
```

3.2.3.2 Parameters

NumOfChars	Number of available characteristics that have been loaded
handle	The client data instance handle, that called the evaluation method (to distinguish between instance, if multiple connections to the server are established)

3.2.4 Statistical Evaluation Synchronisation

This event indicates the client application, that the server has evaluated the data set. After a call to the evaluation-methods, the client application has to wait until it receives this call. Before this event occurs, requests for graphics or numerical results won't be successful.

3.2.4.1 IDL Syntax

```
[id(0x00000004)]
HRESULT _stdcall OnEvaluationDone( void );
[id(0x0000000A)]
HRESULT OnEvaluationDoneExt([in] long handle );
```

3.2.4.2 Parameters

handle	The client data instance handle, that called the evaluation method (to distinguish between instance, if multiple connections to the server are established)
--------	---

3.3 Alarms

The interface for getting events from the Q-DAS server is named:

- IQSEVENTS
(see section 3.2 Events)

In the Online System the server sends alarms to the client by using the connection point technology, so the client interested in the online functionalities should implement the IQsEvent interface and react at least at the following methods:

- OnSystemAlarm
Server informs the client, if any system alarm (not enough disk space etc.) has occurred.
- OnStatisticalAlarm
The server informs the client, if any statistical alarm based on the calculation of the last values has happened.

3.3.1 OnSystemAlarm

This indicates the client, that any problem on the system has occurred (e.g. a warning, if a limit of disc space has been reached). This alarm may be invoked after data has been sent to the server.

3.3.1.1 IDL Syntax

```
[id(0x00000006)]
HRESULT _stdcall OnSystemAlarm( [out] unsigned long AlarmCode );
```

3.3.1.2 Parameters

1. AlarmCode
bit coded system alarm

3.3.2 OnStatisticalAlarm

This event informs the client, if a statistical alarm or exception according to the presetted evaluation has occurred. The alarm may be invoked after data has been sent to the server and the server has done a statistical (online) evaluation. Some alarms are only invoked if a sample or subgroup is completed.

3.3.2.1 IDL Syntax

```
[id(0x00000007)]
HRESULT stdcall OnStatisticalAlarm([in] unsigned long Alarm_EW_1,
                                   [in] unsigned long Alarm_EW_2,
                                   [in] unsigned long Alarm_QRK_1,
                                   [in] unsigned long Alarm_QRK_s,
                                   [in] unsigned long Alarm_cp,
                                   [in] unsigned long Alarm_NO_SPC,
                                   [in] long Part_Nr,
                                   [in] long Char_Nr,
                                   [in] long Value_Nr,
                                   [in] long Sample_Nr);

[id(0x0000000B)]
HRESULT OnStatisticalAlarmExt([in] long handle,
                              [in] unsigned long Alarm_EW_1,
                              [in] unsigned long Alarm_EW_2,
                              [in] unsigned long Alarm_QRK_L,
                              [in] unsigned long Alarm_QRK_S,
                              [in] unsigned long Alarm_CP,
                              [in] unsigned long Alarm_NO_SPC,
                              [in] long Part_Nr,
                              [in] long Char_Nr,
                              [in] long Value_Nr,
                              [in] long Sample_Nr );
```

3.3.2.2 Parameters

handle	The client data instance handle, that called the evaluation method (to distinguish between instance, if multiple connections to the server are established)
Alarm_EW_1	This represents a bit coded alarm for the single value alarms
Alarm_EW_2	Reserved for future developments
Alarm_QRL_1	Bit coded alarms for violations of the QCC-location chart
Alarm_QRK_s	Bit coded alarms for violations of the QCC-variation chart
Alarm_CP	Reserverd for future developments
Alarm_NO_SPC	Reserverd for future developments
Part_Nr	the internal part number, which caused the alarm
Char_Nr	The internal characteristic number, which caused the alarm
Value_Nr	The internal value number, which caused the alarm

Sample_Nr	The internal sample or subgroup number, which caused the alarm
-----------	--

3.3.2.3 Statistical Alarm codes

Parameter	AlarmCode	Meaning
Alarm_EW_1	0x0001	Single value beyond x % of upper spec. limit
	0x0002	Single value below x % of lower spec. limit
	0x0004	Single value beyond upper scrap limit
	0x0008	Single value below lower scrap limit
Alarm_QRK_I	0x0001	Sample / subgroup beyond location warning limits
	0x0002	Sample / subgroup below location warning limits
	0x0010	Sample / subgroup beyond location alarm limits
	0x0020	Sample / subgroup below location alarm limits
	0x0100	Sample / subgroup beyond location control limits
	0x0200	Sample / subgroup below location control limits
	0x0400	Run (up) inside location chart
	0x0800	Run (down) inside location chart
	0x1000	Trend (up) inside location chart
	0x2000	Trend (down) inside location chart
	0x4000	Middlethird (above) inside location chart
	0x8000	Middle third (below) inside location chart
Alarm_QRK_s	0x0001	Sample / subgroup beyond variation warning limits
	0x0002	Sample / subgroup below variation warning limits
	0x0010	Sample / subgroup beyond variation alarm limits
	0x0020	Sample / subgroup below variation alarm limits
	0x0100	Sample / subgroup beyond variation control limits
	0x0200	Sample / subgroup below variation control limits
	0x0400	Run (up) inside variation chart
	0x0800	Run (down) inside variation chart
	0x1000	Trend (up) inside variation chart
	0x2000	Trend (down) inside variation chart
	0x4000	Middlethird (above) inside variation chart
	0x8000	Middle third (below) inside variation chart

4 Additional Interfaces

The purpose of these interfaces is, to give the possibility for querying available functionalities and configurations to the client application. This is divided into groups of functionalities / configuration. Each interfaces has an corresponding `Getxxxx` / `Setxxxx` method inside the `IQsstatRemoteControl` interface.

All EnumInterfaces have methods like

- `GetFirstxxxxx`
- `GetNextxxxxxx`

which allows the client application, to build loops for querying and storing the available functions at runtime or inside resources.

Only some of these interfaces are described here. To get further information, see the typelibrary of the server.

4.1 Interface IEnumLanguage

Via the `IEnumLanguage` the Client application gets the names and the keys for the available languages

The corresponding method is :

```
ITqsSTATRemoteControl.SetLanguage
ITqsSTATRemoteControl.ClienConnect
```

4.1.1 Remarks

In newer versions of the server, this interface can be used without an established connection in order to obtain informations, that have to be passed during the `ClientConnectXXX` commands. The connection handle has to be set to 0 to get information about the available modules before the `ClientConnectXXX` methods has been called. This feature is available at Version 8.2.

4.2 Interface IEnumModule

Via the IEnumModule the Client application gets the names and the keys for the available modules.

The corresponding method is :

```
ITqsSTATRemoteControl.SetModule  
ITqsSTATRemoteControl.ClienConnect
```

4.2.1 Remarks

In newer versions of the server, this interface can be used without an established connection in order to obtain informations, that have to be passed during the `ClientConnectXXX` commands. The connection handle has to be set to 0 to get information about the available modules before the `ClientConnectXXX` methods has been called. This feature is available at Version 8.2.

4.3 IEnumEvaluationStrategy

Via the IEnumEvaluationStrategy the client application can retrieve the keys names for the available evaluation strategies.

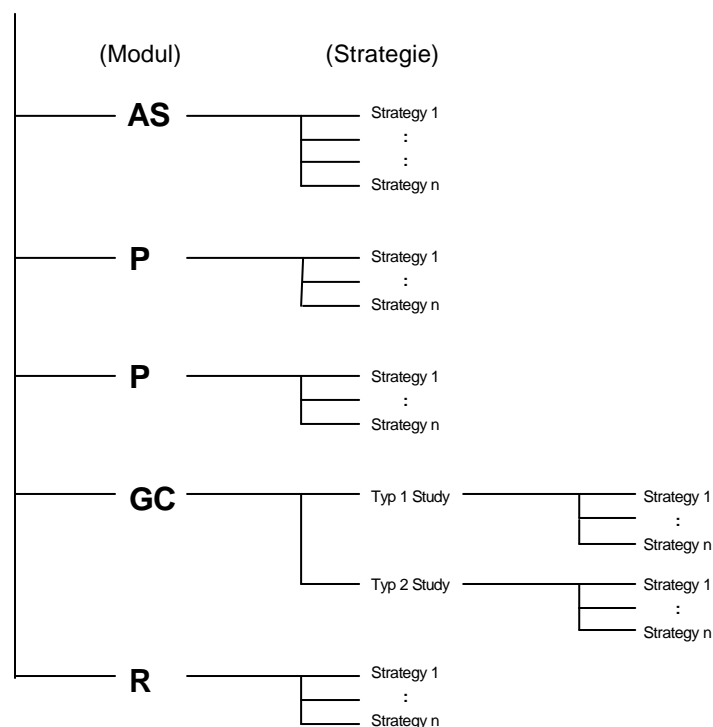
The corresponding method to activate an existing strategy is

```
ITqsSTATRemoteControl.SetStrategy
```

4.3.1 Dependencies

Which strategies are available, depends on the qs-STAT module. The modules process capability (moduleKey = 20) and procella / Online server (moduleKey = 26) both have the same evaluation strategies.

Module



4.4 Interface IEnumGraphic

Via the IEnumGraphic the Client application can get the names and the keys for the available graphics. Which graphics are available depends on the preset module.

The corresponding methods are :

```
ITqsSTATRemoteControl.GetGraphic  
ITqsSTATRemoteControl.GetGraphicExt
```

4.4.1 Dependencies

The availability of graphic depends on the selected qs-STAT module.

4.5 Interface IEnumStatResults

Via the IEnumReports the Client application can get the names and the keys for the available numerical results. Which numerical results are available depends on the preset module.

The corresponding methods are :

```
ITqsSTATRemoteControl.GetStatResult  
ITqsSTATRemoteControl.GetStatResultExt
```

4.5.1 Dependencies

The availability of statistical results depends on the selected qs-STAT module.

4.6 Interface IEnumReports

Via the IEnumReports the Client application can get the names and the keys for the available reports. Which reports are available depends on the preset module.

The corresponding methods are :

```
ITqsSTATRemoteControl.GetReport  
ITqsSTATRemoteControl.PrintReport  
ITqsSTATRemoteControl.PrintReportExt  
ITqsSTATRemoteControl.GetReportPages
```

4.6.1 Dependencies

The availability of statistical results depends on the selected qs-STAT module.

4.7 Interface IEnumCatalogue

Via the IEnumCatalogue the Client application can retrieve a list of the activated catalogues and their entries.

The methods:

- GetFirstCatalogue
- GetNextCatalogue

provide a list of available catalogues to the client application. Each catalogue is represented by a numerical key, that can be used to query the available entries inside this catalogue.

The methods:

- GetFirstCatalogEntry
- GetNextCatalogEntry

provide an access to the entries of a specific catalog. The catalog key can be retrieved by calls to GetFirst/GetNextCatalogue

The methods:

- GetFirstSubCatalogue
- GetNextSubCatalogue

provide an access to defined subgroups / subsets of a specific catalogue.

4.8 Interface IEnumFieldName

This interface allows the client application, to query the available fields on the qs-STAT parts-, characteristic- and value level. The methods are:

- GetFirstPartField
K1000-K1999
- GetNextPartField
K1000-K1999
- GetFirstCharacteristicField
K2000-K2999, K8000-K8999
- GetNextCharacteristicField

- K2000-K2999, K8000-K8999
- GetFirstValueField
K0001-K0099
- GetNextValueField
K0001-K0099

4.9 Interface IEnumDBSelectionName

This interface allows the client application to ask for the stored database queries. The queries are stored according to the current user, so only these queries are visible for the client. If the client wants to read from the Q-DAS database, it's recommended to use one of these stored queries.

The following methods are available:

- GetFirstDBSelection
Returns the first stored database query for the current user
- GetNextDBSelection
Returns the next available database query
- SkipDBSelection
Skips a specific number of queries
- CloseDBSelection
This method has to be called when the quest for available queries is finished.

4.10 Interface IQsstatData

This Interface offers functions which can be used to manipulate qs-STAT data.

```
interface IQsstatData: IDispatch
[
    uuid(9BFCB1E2-BB6E-4E62-947F-C0BB7108AC68),
    version(1.0),
    dual,
    oleautomation
]
```

4.10.1 SetKey

This function allows to change part, characteristic or value data.

4.10.1.1 IDL Syntax

```
[id(0x00000001), helpstring("Runtime Access to loaded data")]
HRESULT _stdcall SetKey (
                                [in]          long handle,
                                [in]          VARIANT PartNr,
                                [in]          VARIANT CharNr,
                                [in]          VARIANT GroupNr,
                                [in]          VARIANT ValueNr,
                                [in]          long Key,
                                [in]          long SubKey,
                                [in]          BSTR Data,
                                [in]          long CharRange);
```

4.10.1.2 Parameters

handle	Qs-STAT handle received by the client connect command
PartNr	Specifies the part number
CharNr	specifies the characteristic number
GroupNr	specifies the group number
ValueNr	specifies the value number
Key	specifies the field to be changed (KXXXX).
SubKey	Not yet used
Data	The new field value
CharRange	Specifies the range of characteristics to be affected, as described below.

CharRange		
\$00	WKCRM_NO_RANGE_C	No range specified
\$01	WKCRM_ALL_CHAR_C	All characteristics

\$02	WKCRM_ALL_CHAR_PART_C	All characteristics for a specified part
\$04	WKCRM_LIST_C	All listed characteristics
\$10	WKCRM_SINGLE_CHAR_C	One specified characteristic

4.10.1.3 Remarks:

Online Server: The data is evaluated immediately without a call to EvaluateXXXX – methods and the alarm events won't be raised.

4.10.2 SortValues

This function allows to sort values in qs-STAT.

4.10.2.1 IDL Syntax

```
[id(0x00000002)]
HRESULT _stdcall SortValues (
    [in] long handle,
    [in] VARIANT PartNr,
    [in] VARIANT CharNr,
    [in] VARIANT GroupNr,
    [in] BSTR OrderBy,
    [in] long CharRange);
```

4.10.2.2 Parameters

Handle	Qs-STAT handle received by the client connect command
PartNr	Specifies the part number (Reserve)
CharNr	Specifies the characteristic number (Reserve)
GroupNr	Specifies the group number (Reserve)
OrderBy	A list of fields to order the values by, e.g. "K0006 DESC, K0004 ASC" to sort all values by descending batch numbers and, in case of equal batch numbers, by ascending date and time.
CharRange	Specifies the range of characteristics to be affected, as described below.

4.10.2.3 Remarks

(The specification of PartNr, CharNr, GroupNr and CharRange are not yet supported, SortValues sorts the values of all loaded characteristics instead.)

4.10.3 DeleteValues

This function allows delete measurement values from the loaded data set

4.10.3.1 IDL Syntax

```
[id(0x00000003), helpstring("Delete values at loaded data during runtime")]
HRESULT _stdcall DeleteValues(           [in] long    handle,
                                         [in] VARIANT PartNr,
                                         [in] VARIANT CharNr,
                                         [in] VARIANT ValueNr,
                                         [in] VARIANT NrOfValues,
                                         [in] long    CharRange );
```

4.10.3.2 Parameters

handle	Qs-STAT handle received by the client connect command
PartNr	Specifies the part number
CharNr	specifies the characteristic number
GroupNr	specifies the group number
ValueNr	specifies the value number
NrOfValues	specifies the number of values, that shall be deleted
CharRange	Specifies the range of characteristics to be affected, as described below.

CharRange		
\$00	WKCRM_NO_RANGE_C	No range specified
\$01	WKCRM_ALL_CHAR_C	All characteristics
\$02	WKCRM_ALL_CHAR_PART_C	All characteristics for a specified part
\$04	WKCRM_LIST_C	All listed characteristics
\$10	WKCRM_SINGLE_CHAR_C	One specified characteristic

4.10.4 DFDCharNr2PartCharNr

This function converts a characteristic number from the DFD or DFQ file (incremental position) towards the logical characteristic number necessary for calls to the GetXXX methods. It can be used in combination with the OpenFileXXX or TransmitFileXXX methods only. This method might be helpful when using other s than flat characteristic structures or multiple parts per DFD / DFQ file or stream.

4.10.4.1 IDL Syntax

```
[id(0x00000004), helpstring("Converts a given DFD char. position to logical
part and characteristic numbers")]
HRESULT _stdcall DFDCharNr2PartCharNr( [in] long    handle,
```

```
[in] long    DFDCharNr,
[out] long * PartNr,
[out] long * CharNr );
```

4.10.4.2 Parameters

handle	Qs-STAT handle received by the client connect command
DFDCharNr	The characteristic position inside the file
PartNr	Return value: logical part number for addressing a characteristic
CharNr	Return value: logical characteristic number for addressing a characteristic (in combination with the part number)

Remarks:

Only in combination with data set based on the Q-DAS Ascii-Transferformat

4.10.5 PartCharNr2DFDCharNr

This function converts a characteristic number from the DFD or DFQ file (incremental position) towards the logical characteristic number necessary for calls to the GetXXX methods. It can be used in combination with the OpenFileXXX or TransmitFileXXX methods only. This method might be helpful when using other s than flat characteristic structures or multiple parts per DFD / DFQ file or stream.

4.10.5.1 IDL Syntax

```
[id(0x00000005), helpstring("Converts a given logical part and
characteristic numbers to DFD char. positions")]
HRESULT _stdcall PartCharNr2DFDCharNr(
    [in] long    handle,
    [in] long    PartNr,
    [in] long    CharNr,
    [out] long * DFDCharNr );
```

4.10.5.2 Parameters

Handle	Qs-STAT handle received by the client connect command
PartNr	logical part number for addressing a characteristic
CharNr	logical characteristic number for addressing a characteristic (in combination with the part number)
DFDCharNr	Return value: the characteristic position inside the file

Remarks:

Only in combination with data set based on the Q-DAS Ascii-Transferformat

4.11 Interface IEnumUser

This interface can be used to query the present users in the server's user administration. It returns the names of all available users to the client. This interfaces is usable without an established connection.

The corresponding method, that can be feeded with that information are:

`ITqsSTATRemoteControl.ClientConnectXXX`

4.11.1 IDL Syntax

```
[id(0x00000001)]
HRESULT      _stdcall GetAllUsers(  [out] VARIANT *    UserList );
    Returns all available users in an array of strings
[id(0x00000002)]
HRESULT      _stdcall GetFirstUser( [out] BSTR *      UserName );
    Returns the first available user in a string
[id(0x00000003)]
HRESULT      _stdcall GetNextUser(  [out] BSTR *      UserName );
    Returns the next available user in a string
```

4.11.2 Parameters

UserList	Return value: all usernames in form of an array of strings
UserName	Return value: username in form of a string

4.11.3 Remarks

This interface is available at newer versions only (Version > 8.2)

5 DCOM Security Aspects

If the statistical server runs as a DCOM server, a few security aspects have to be considered. Since the clients have to implement a connection point, access permissions for the server on the client machine have to be given. This can be done by using the

DCOMCNFG tool,

delivered with the operating system. The easiest way is to give general access for everyone to start and use objects at the default permissions. If the username of the server PC is known, these rights may be given also for that specific user directly.

This has an effect on the following keys inside the registry:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Ole]
"EnableDCOM"="Y"
"DefaultLaunchPermission"=...
"DefaultAccessPermission"=...
```