q-das@q-das.de • Tel. 06201/3941-0

q-das@q-das.de • Tel. 06201/3941-0

q-das@q-das.de • Tel. 06201/3941-0

g-das@q-das.de • Tel. 06201/3941-0

1 Introduction

The new Q-DAS web service, which is released in Version ME 8 of Q-DAS' product suite, is to replace the previous "WebServiceQsSTAT" web service. The old web service consisted of a DLL interface that had to connect itself to qs-STAT via qs-STAT's COM interface, which usually produced a hard job of trouble-shooting, and it translated qs-STAT's methods and parameters into C# methods and their parameters. The new web service consists of a DLL file which itself contains the complete functionality of qs-STAT, so that it has become obsolete to connect to qs-STAT via COM. The syntax of the methods and their parameters were kept the same as in the old web service as far as possible, but the more complex parameters, the type of which was **`System.Xml.XmlElement`**, had to be replaced by their **`string`** equivalents.

2 Getting started

If you do not use qs-STAT's or M-QIS' web designer, you can use Microsoft's Internet Information Services (IIS) to create a virtual web directory and copy the file "QdasWebService.dll", which you can find in the "…\Web\QsStatWebService" folder of your qs-STAT installation, into the virtual directory's local folder. Additionally, an INI file named "qs-STAT-loc.ini" has to be created in the same folder and with the following contents:

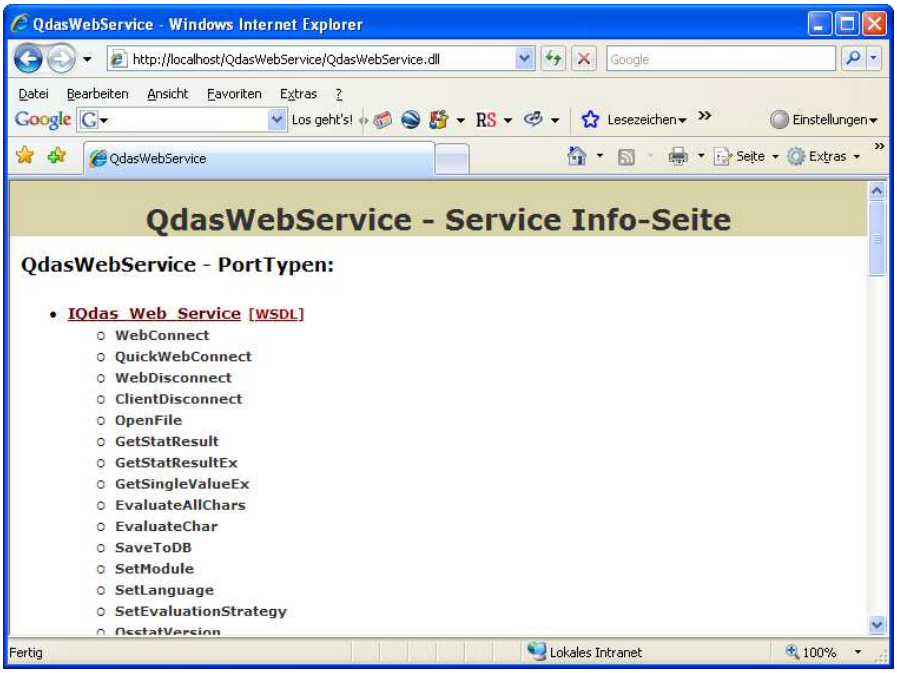
```
[Paths]
qs_stat_exe_path=< BIN path of your qs-STAT installation >
qs_stat_ini_file=< the path and filename of your QSSTAT2000.IN I>
webservice_logfile=< the path and name of a log file (optional) >
logfile_level=0|1|2 < optional:
                      0 = no log
                      1 = a few log file messages
                      2 = more log file messages >
```

This file supplies the web service with the information where to find all of its settings. Now you can open a browser and navigate to the DLL file in your new virtual directory. Usually this should be

`http://localhost/<new virtual directory name>/QdasWebService.dll`.

If your computer does not accept "localhost", you can try to use ist IP address instead. Now you should get something like this:

q-das@q-das.de • Tel. 06201/3941-0



Eventually you have to add “QdasWebService.dll” to the “Web Service Extensions” list in ISS (if available) to receive the image above.

Now you can start writing your web application with an appropriate development environment. You will have to add a reference to your project which creates the connection to the web service; in Visual Web Developer 2005, e.g., you can achieve this by right-clicking on your project (in the tree view), selecting the “Add web reference…” menu item and navigating to the same address as above. As a result you should get a new class “`IQdas_Web_Serviceservice`” in the namespace “`localhost`”.
Because variables and objects of web applications usually are not persistent, you will have to define one or more classes of containing objects which keep the persistent data, e.g.:

```
public class SessionInfo : System.Object
{
      public localhost.IQdas_Web_Serviceservice ws = null;
      public int QsStat_Handle = 0;
}
```

This object should be created once, saved as a session object after each change and loaded at the beginning of the `Page_Load` method:

```
Info = new SessionInfo();

Session["SessionInfo"] = Info;

Info = (SessionInfo)Session["SessionInfo"];
```

Once the web service object has been created, you can call its methods:

```
Info.ws = new localhost.IQdas_Web_Serviceservice();
Info.ws.WebConnect(20, 44, "SuperUser", "superuser",
      "0.0.0.0", out Info.QsStat_Handle );
```


q-das@q-das.de • Tel. 06201/3941-0

The `QsStat_Handle` output parameter will be needed as an input parameter in most of the web service's methods as it defines the connection to persistent data on the service side.

When the session is finished, the session-related resources on the service should be released:

```
Info.ws.ClientDisconnect(Info.QsStat_Handle);
```

This should be included in the `Session_End` event of `Global.asax.cs`:

```
protected void Session_End(Object sender, EventArgs e)
{
      if (Session["SessionInfo"] != null)
      {
            SessionInfo Info = (SessionInfo)Session["SessionInfo"];
            Info.ws.ClientDisconnect(Info.QsStat_Handle);
            Info.ws = null;
      }
}
```


g-das@q-das.de • Tel. 06201/3941-0

3 List of the Commands Supported by Q-DAS Web Service

All commands listed below are members of `localhost.IQdas_Web_Serviceservice`.

3.1 Connection Commands

3.1.1 WebConnect

This method is needed to establish a qs-STAT session, the representation of which is the returned handle parameter.

Syntax:

```
public int WebConnect(  int ModuleID,
                        int LanguageID,
                        string UserID,
                        string UserPwd,
                        string ClientAddress,
                        out int Handle);
```

Parameters:

`ModuleID`	Startup module key for qs-STAT: 1: Reliability analysis; 10: Sample analysis; 20: Process capability; 30: Gage capability; 80: Long term analysis
`LanguageID`	Startup language key for qs-STAT: 44: English; 49: German;
`UserID`	qs-STAT User's connection name
`UserPwd`	qs-STAT User's connection password
`ClientAddress`	IP address of the Client PC, may be used to count the number of different connected users and compare it with the total number of qs-STAT licenses
`Handle`	This is the returned qs-STAT handle which identifies the qs-STAT session and has to be used as input parameter in all other commands
`returns`	0 on success

3.1.2 QuickWebConnect

This method is for testing purposes only. It creates a connection similar to WebConnect, but with the following parameters: `ModuleID = 20; LanguageID = 49; UserID = "SuperUser"; UserPwd = "SuperUser", ClientAddress = "0.0.0.0"`. It returns the qs-STAT handle.

Syntax:

```
public int QuickWebConnect();
```

Parameters:

`returns`	qs-STAT handle on success, otherwise 0

q-das@q-das.de • Tel. 06201/3941-0

3.1.3 WebDisconnect

This method closes a qs-STAT session which is represented by the handle parameter.

Syntax:
```
public int WebDisconnect( int Handle);
```

Parameters:

`Handle`	This is the handle of the qs-STAT session to close.
`returns`	0 on success

3.1.4 ClientDisconnect

This method is just a synonym for WebDisconnect.

Syntax:
```
public int ClientDisconnect( int Handle);
```

Parameters:

`Handle`	This is the handle of the qs-STAT session to close.
`returns`	0 on success

public int QsstatVersion(int Handle, out string VersionNr);

3.1.5 QsstatVersion

This method returns the version number of qs-STAT.

Syntax:
```
public int QsstatVersion(     int Handle,
                              out string VersionNr );
```

Parameters:

`Handle`	This is the handle of the qs-STAT session.
`VersionNr`	The returned version number
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.2 Text Command

3.2.1 GetQSStatText

This method returns a text from qs-STAT's text database.

Syntax:

```
public int GetQSStatText(     int Handle,
                              int TextGroup,
                              int TextKey,
                              int TextSubKey,
                              int SingularPlural,
                              out string Text );
```

Parameters:

`Handle`	This is the handle of the qs-STAT session.
`TextGroup`	Group key of the text in the database
`TextKey`	Key of the text in the database
`TextSubKey`	Subkey of the text in the database
`SingularPlural`	0 on singular, 1 on plural
`Text`	The returned text from database
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.3 Commands to handle Session settings

All pairs of `GetFirstXXX` / `GetNextXXX` methods described in this chapter can be used to get enumerations of certain items. This is achieved by calling the `GetFirstXXX` method once, and on success (return code = 0) calling the `GetNextXXX` method as long as it returns 0 (zero).

3.3.1 GetFirstUser

Together with `GetNextUser` this Method can be used to get an enumeration of all qs-STAT users. Both methods do *not* need a handle parameter, so that a list of users may be received *before* a connection to a qs-STAT session is performed.

Syntax:
```
public int GetFirstUser(out string UserName);
```

Parameters:

`UserName`	returns user name
`returns`	0 on success

3.3.2 GetNextUser

See above (`GetFirstUser`).

Syntax:
```
public int GetNextUser(out string UserName);
```

Parameters:

`UserName`	returns user name
`returns`	0 on success

3.3.3 GetFirstModule

Together with `GetNextModule` this method can be used to get an enumeration of all available qs-STAT modules.

Syntax:
```
public int GetFirstModule(    int Handle,
                              out int Module,
                              out string ModuleName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Module`	returns the module key
`ModuleName`	returns the module name
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.3.4 GetNextModule

See above (`GetFirstModule`).

Syntax:

```
public int GetNextModule(     int Handle,
                              out int Module,
                              out string ModuleName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Module`	returns the module key
`ModuleName`	returns the module name
`returns`	0 on success

3.3.5 GetFirstModuleExt

This method, and also `GetNextModuleExt`, is only to be used by applications made by Q-DAS.

Syntax:

```
public int GetFirstModuleExt( int int QDasTool,
                              int QdasID,
                              out int Module,
                              out string ModuleName );
```

Parameters:

`QDasTool`	internal key of a tool made by Q-DAS
`QDasID`	internal authentication key
`Module`	returns the module key
`ModuleName`	returns the module name
`returns`	0 on success

3.3.6 GetNextModuleExt

See above (`GetFirstModuleExt`).

Syntax:

```
public int GetNextModuleExt(  int QDasTool,
                              int QdasID,
                              out int Module,
                              out string ModuleName );
```

Parameters:

`QDasTool`	internal key of a tool made by Q-DAS
`QDasID`	internal authentication key
`Module`	returns the module key
`ModuleName`	returns the module name
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.3.7 SetModule

This method changes the current module setting of the qs-STAT session.

Syntax:
```
public int SetModule(   int Handle,
                        int Module );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Module`	new module key
`returns`	0 on success

3.3.8 GetFirstLanguage

Together with `GetNextLanguage` this method can be used to get an enumeration of all available qs-STAT modules.

Syntax:
```
public int GetFirstLanguage(  int Handle,
                              out string language_str,
                              out string language_shortStr,
                              out int language );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`language_str`	returns the language name
`language_shortStr`	returns the language abbreviation
`language`	returns the language key
`returns`	0 on success

3.3.9 GetNextLanguage

See above (`GetFirstLanguage`).

Syntax:
```
public int GetNextLanguage(   int Handle,
                              out string language_str,
                              out string language_shortStr,
                              out int language );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`language_str`	returns the language name
`language_shortStr`	returns the language abbreviation
`language`	returns the language key
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.3.10 GetFirstLanguageExt

This method is only to be used by applications made by Q-DAS.

Syntax:

```
public int GetFirstLanguageExt(     int int QDasTool,
                                    int QdasID,
                                    out string language_str,
                                    out string language_shortStr,
                                    out int language );
```

Parameters:

`QDasTool`	internal key of a tool made by Q-DAS
`QDasID`	internal authentication key
`language_str`	returns the language name
`language_shortStr`	returns the language abbreviation
`language`	returns the language key
`returns`	0 on success

3.3.11 SetLanguage

This method changes the language of the current qs-STAT session.

Syntax:

```
public int SetLanguage( int Handle,
                        int Language );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Language`	new language key
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.3.12 GetFirstEvaluationStrategy

Together with `GetNextEvaluationStrategy` this method can be used to get an enumeration of all available qs-STAT evaluation strategies for a given module.

Syntax:

```
public int GetFirstEvaluationStrategy(    int Handle,
                                          int Modul,
                                          int Study,
                                          out int StratNr,
                                          out string StratName);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Modul`	module key
`Study`	study key
`StratNr`	returns the strategy key
`StratName`	returns the strategy name
`Returns`	0 on success

3.3.13 GetNextEvaluationStrategy

See above (`GetFirstEvaluationStrategy`).

Syntax:

```
public int GetNextEvaluationStrategy (    int Handle,
                                          int Modul,
                                          int Study,
                                          out int StratNr,
                                          out string StratName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Modul`	module key
`Study`	study key
`StratNr`	returns the strategy key
`StratName`	returns the strategy name
`Returns`	0 on success

3.3.14 GetDefaultEvaluationStrategy

This method returns the default evaluation strategy for a given module.

Syntax:

```
public int GetDefaultEvaluationStrategy ( int Handle,
                                          int Modul,
                                          out int StratNr,
                                          out string StratName );
```


q-das@q-das.de • Tel. 06201/3941-0

Parameters:

`Handle`	the handle of the qs-STAT session
`Modul`	module key
`StratNr`	returns the strategy key
`StratName`	returns the strategy name
`Returns`	0 on success

3.3.15 SetEvaluationStrategy

This method changes the current evaluation strategy of the qs-STAT session.

Syntax:

```
public int SetEvaluationStrategy(   int Handle,
                                    int Study,
                                    int StratNr );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Study`	study key
`StratNr`	returns the strategy key
`Returns`	0 on success

3.3.16 GetLTTree

In module long term analysis (LT) this method returns a tree structure showing the structure of all available stored results, separated by the module in which the evaluation took place, the evaluation strategy and the type of subdivision by additional data. The tree structure is returned as the text representation of an XML element.

Syntax:

```
public int GetLTTree(   int Handle,
                        out string LTTree_XMLStr );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`LTTree_XMLStr`	returns the tree structure (see below)
`Returns`	0 on success

XML structure (example):

```
<Root>
      <Module Key="20" Name="Process Capability Analysis">
            <EvalStrat Key="23" Name="Q-DAS 1 - Part">
                  <LTFields Fields="K4(0);" Names="Division: year"/>
                  <LTFields Fields="K4(1);" Names="Division: month"/>
                  <LTFields Fields="K4(1);K10;"
                              Names="Division: month, Machine number"/>
            </EvalStrat>
      </Module>
</Root>
```


q-das@q-das.de • Tel. 06201/3941-0

The root node contains a node that describes the module in which the evaluation has been performed, identified by the attribute `Key` and having a readable description in the attribute `Name`. There may be more than one module node. Each module node contains at least one `EvalStrat` node, showing the evaluation strategy under which the evaluation has been performed, also identified by the `Key` attribute and having a description in in the `Name` attribute. The `LTFields` nodes describe how the data have been separated by additional data before performing the evaluation. In this node the `Fields` attribute contains a list of K fields used to separate the measurement values – in case of K4 (date) the number in brackets describes the way of date separation: 0 – by year, 1 – by month, 2 – by week, 3 – by day, 4 – by hour; 11 – by quarter. The `Names` attribute just provides a readable form of the separation field list.
In the above example there are evaluation results available for the module PC (20), the evaluation strategy No. 23 with three different ways of separation – first by years, then by months, finally a separation by months and a sub-separation by machines.

3.3.17 SetLTEnvironment

In module long term analysis (LT) this method is used to set one of the parameter combinations found with the `GetLTTree` method.

Syntax:

```
public int SetLTEnvironment(  int Handle,
                              int LTModule,
                              int LTStrategy,
                              string LTAufteilung);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`LTModule`	the module for which data are to be found (to be taken from the `Module.Key` attribute in `GetLTTree`)
`LTStrategy`	the evaluation strategy for which data are to be found (to be taken from the `EvalStrat.Key` attribute in `GetLTTree`)
`LTAufteilung`	the separation key list for which data are to be found (to be taken from the `LTFields.Fields` attribute in `GetLTTree`)
`Returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.3.18 SetProperty

This method can be used to set certain session properties listed below.

Syntax:

```
public int SetProperty( int Handle,
                        int Prop_Key,
                        string Prop_Value);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Prop_Key`	property key (listed below)
`Prop_Value`	the value the property is to be set to
`Returns`	0 on success

Property list:

Property key	Intention	Property value
4001	To set the evaluation module in long term analysis	A module key, e.g. 20
4002	To set the evaluation strategy in long term analysis	An evaluation strategy key number
4003	To load a long term analysis configuration	The name of an LT configuration
5001		
5002		
6001	To set a database connection	An UDL file name
6100	To abort reading from database	1
7001	To export loaded data as QML	The path to export to

g-das@q-das.de • Tel. 06201/3941-0

3.4 Data File Commands

3.4.1 OpenFile

This method loads a .dfd or .dfq file to qs-STAT.

Syntax:

```
public int OpenFile(    int Handle,
                        string filename);
```

Parameters:

`Handle`	The handle of the qs-STAT session.
`filename`	The path and filename of the file to open.
`returns`	0 on success

3.4.2 SaveToFile

This method saves the data of the qs-STAT session to a .dfd or .dfq file.

Syntax:

```
public int SaveToFile(  int Handle,
                        string filename);
```

Parameters:

`Handle`	The handle of the qs-STAT session.
`filename`	The path and filename of the file to save.
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.5 Database Commands

3.5.1 GetFirstQueryName

Together with `GetNextQueryName` this method can be used to get an enumeration of all queries stored in qs-STAT (in the configuration database) available to the current user under the current module.

Syntax:

```
public int GetFirstQueryName( int Handle,
                              out string QueryName);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryName`	the name of the stored query
`returns`	0 on success

3.5.2 GetNextQueryName

See above (`GetFirstQueryName`).

Syntax:

```
public int GetNextQueryName(  int Handle,
                              out string QueryName);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryName`	the name of the stored query
`returns`	0 on success

3.5.3 LoadQuery

This method loads the specified stored query into the qs-STAT session. The returned `QueryHandle` parameter identifies that query in the session. The handle is needed in all commands that work on the query. When the query is no longer in use it should be released by calling the `FreeQuery` method.

Syntax:

```
public int LoadQuery(   int Handle,
                        string QueryName,
                        out int QueryHandle);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryName`	the name of the stored query
`QueryHandle`	returns the query handle to be used in further query commands
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.5.4 LoadQueryExt

Like `LoadQuery` this method loads a stored query. Additionally it returns a parts / characteristics list.

Syntax:

```
public int LoadQueryExt(      int Handle,
                              string QueryName,
                              out int QueryHandle,
                              out string Part_Char_List);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryName`	the name of the stored query
`QueryHandle`	returns the query handle to be used in further query commands
`Part_Char_List`	returns a parts / characteristics list
`returns`	0 on success

XML structure (example):

```
<PartCharList>
      <Part key="14">
            <Char key="1">
            <Char key="3">
            <Char key="6">
      </Part>
</PartCharList>
```

In the `Part` and `Char` nodes the `key` attribute denotes the database key of the part or characteristic. A `Part` node without `Char` nodes is equivalent to a part without restrictions to the characteristics, i.e. a part with all its characteristics.

SaveQuery

This method saves a query in qs-STAT's configuration database under the current user in the current module under the given name.

Syntax:

```
public int SaveQuery(   int Handle,
                        string QueryName,
                        int QueryHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryName`	the name of the query is to be stored under
`QueryHandle`	the query handle
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.5.5 CreateQuery

This method creates a new database query object in the qs-STAT session. Like `LoadQuery` (see above) it returns a handle.

Syntax:

```
public int CreateQuery( int Handle,
                        out int QueryHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	returns the query handle
`returns`	0 on success

3.5.6 CreateQueryFromXML

Like `CreateQuery` this method creates a new database query object in the qs-STAT session. In contrast to `CreateQuery`, which creates a new empty query from scratch, the query object in this case gets a complete description of its properties (e.g. filters, part list).

Syntax:

```
public int CreateQueryFromXML(      int Handle,
                                    string QueryXMLString,
                                    out int QueryHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryXMLString`	the string representation of an XML element which contains the query's properties
`QueryHandle`	returns the query handle
`returns`	0 on success

3.5.7 AddFilterToQuery

This method adds a filter object (identified by the filter handle) to the query object identified by the query handle.

Syntax:

```
public int AddFilterToQuery(  int Handle,
                              int QueryHandle,
                              int FilterHandle,
                              byte QueryLevel,
                              int Part_Key,
                              int Char_Key );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`FilterHandle`	the filter handle

q-das@q-das.de • Tel. 06201/3941-0

`QueryLevel`	0 – the filter is used to filter parts 1 – the filter is used to filter characteristics 2 – the filter is used to filter measurement values
`Part_Key`	0 – the filter is applied globally > 0 – the filter is only applied to the specified part (only with `QueryLevel` 1 & 2)
`Char_Key`	0 – the filter is applied globally > 0 (and `Part_Key` > 0) the filter is only applied to the specified characteristic (only with `QueryLevel` 2)
returns	0 on success

3.5.8 AddUserGroupFilterToQuery

If the connected qs-STAT user belongs to a group a restricting part filter is assigned to, this method adds the restricting part filter to the query object identified by the query handle. The method has to be called after normal part filters have been added; if there is already a part filter existing in the query, it is merged with the user group filter.

Syntax:

```
public int AddUserGroupFilterToQuery (    int Handle,
                                          int QueryHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
returns	0 on success

3.5.9 AddPartCharacteristicListToQuery

This method adds a parts / characteristics list to the query object identified by the query handle.

Syntax:

```
public int AddPartCharacteristicListToQuery (   int Handle,
                                                int QueryHandle,
                                                string Part_Char_List );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Part_Char_List`	the string representation of an XML element containing the list of parts and characteristics
returns	0 on success

The structure of the XML element is described above in `LoadQueryExt`.

3.5.10 AddSortToQuery

g-das@q-das.de • Tel. 06201/3941-0

This method tells the query in which order to return its results.

Syntax:
```
public int AddSortToQuery(    int Handle,
                              int QueryHandle,
                              int SortKey,
                              byte Direction);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`SortKey`	the "K" code of the fields the output is to be sorted by (it is possible to call the method more than once, with different sort keys)
`Direction`	0 – ascending 1 - descending
`returns`	0 on success

3.5.11 SetQueryProperty

This method is used to set certain properties of a query object (see list below).

Syntax:
```
public int SetQueryProperty(  int Handle,
                              int QueryHandle,
                              int PropertyKey,
                              string PropertyValue );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`PropertyKey`	the property key (see list below)
`PropertyValue`	the value of the property
`returns`	0 on success

Properties:

`PropertyKey`	**`PropertyValue`**
5001	Path and name of a log file. If empty, no log is written.
5010	1 – data of all parts are joined to one part. Characteristics of the parts are joined if they are equal in K2001 and K2002. 0 – no joining
5011	1 – data of all characteristics are joined to one characteristic 0 – no joining
5012	(together with 5010:) database key of the part under which all data are joined
5013	(together with 5010:) 1 – measurement values are sorted among all data 0 – measurement values are sorted inside the parts they belong to, but the outer order is an image of the parts which are joined
5014	(together with 5010:) a list of characteristic fields (e.g. "2001,2003") is used instead of the default fields (K2001, K2002) on joining characteristics

q-das@q-das.de • Tel. 06201/3941-0

5015	1 – If the query returns two parts, they are joined to one part, and differences between the measurement values are calculated. 0 – switch this option off
5016	(together with 5010:) a list of part fields (e.g. “1001,1002”) – only parts which are equal by this field list are joined together
5017	1 – subordinate parts which have a master part inside the query are joined together 0 – off
5018	(together with 5010:) 1 – Characteristics are not joined
5020	(when using the “automatic query”:) 1 – characteristics appear on the “leaf” branches of the tree structure, additional data divisions appear as group nodes 2 – characteristics appear on the “leaf” branches of the tree structure, additional data divisions appear as part nodes 0 – characteristics make the main branches of the tree structure, additional data divisions appear as subordinate characteristics
5030	(only with 5020 = 0:) 1 – measurement values are duplicated, they appear in the main branch characteristics and also in the leaves of the tree structure
5050	6, 14, 16, 17, 53..60 – sort all measurement values by a serial number (K0006 or other additional data fields, only text fields possible) and fill the data structure up with empty measurement values in characteristics where a serial number is missing -1 – sort all measurement values by database value key and fill the data structure up with empty measurement values in characteristics where a key number is missing 0 - off
5051	(together with 5050:) 1 – add additional data to the filled-up empty measurement values as far as possible 0 – off
5052	(together with 5050:) 1 – fill up measurement values across part limits 0 – off
5060	(internal use)
5061	(internal use)
5062	(internal use)
5080	(internal use)
5081	(internal use)
5090	1 – measurement values are not loaded 0 – measurement values are loaded (default behaviour)
5095	1 – when having a parts / characteristics list that contains parts without characteristics, the characteristics of these parts are not loaded 0 – when a part key is given in the parts / characteristics list, without having characteristics information, all characteristics of the part are loaded (default behaviour)
5100	(internal use)
5101	(internal use)
5110	(internal use)
5111	(internal use)
5120	An integer number > 0 – Characteristics are loaded only if they have at least this number of measurement values 0 – All Characteristics are loaded (default behaviour)
5121	(together with 5120:) 1 – If non-loading of characteristics would result in

g-das@q-das.de • Tel. 06201/3941-0

	empty groups superior to those (absent) characteristics, these groups are not loaded either. 0 – All superior groups are loaded (default behaviour).
5130	1 – If the filter conditions exclude groups which have child characteristics that are not excluded, these parent groups are re-included to keep the tree structure. 0 – If groups are excluded, their child characteristics will be attached directly to the part node (default behaviour).
5131	1 – Child characteristics are automatically loaded if their groups are loaded 0 – Child characteristics are loaded according to the filtering conditions (default behaviour)
5132	1 – If groups are excluded, their children are also excluded 0 – If groups are excluded, their child characteristics will be attached directly to the part node (default behaviour).
5140	(internal use)
5141	(internal use)
5142	(internal use)
5150	(internal use)

g-das@q-das.de • Tel. 06201/3941-0

3.5.12 GetFirstPartQuery

This method retrieves information about the first part the query can return. Together with `GetNextPartQuery` it can be used to create a list of parts.

Syntax:

```
public int GetFirstPartQuery( int Handle,
                              int QueryHandle,
                              string KFieldList,
                              out string KResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`KFieldList`	the string representation of an XML element which contains a list of the K-fields which have to be returned
`KResultList`	returns an XML string containing the database key and the field contents of the first part defined by `KFieldList`
returns	0 on success

XML structure of `KFieldList` (example):

```
<FieldList>
      <Field key="1001" />
      <Field key="1002" />
</FieldList>
```

XML structure of returned `KResultList` (example):

```
<KFieldList key="1">
      <Field value="12345"/>
      <Field value="Crankshaft"/>
</KFieldList>"
```

3.5.13 GetNextPartQuery

After a successful call of `GetFirstPartQuery` further parts can be retrieved by calling this method as long as it returns 0 (zero).

Syntax:

```
public int GetNextPartQuery(  int Handle,
                              int QueryHandle,
                              out string KResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`KResultList`	returns an XML string containing the database key and the field contents of the part defined by the `KfieldList` of the previous call to `GetFirstPartQuery`
returns	0 on success

XML structure of returned `KResultList`: See `GetFirstPartQuery`.

q-das@q-das.de • Tel. 06201/3941-0

3.5.14 SkipPartQuery

This method does the same as `GetNextPartQuery`, but skips a given number of parts before returning part data.

Syntax:

```
public int SkipPartQuery(     int Handle,
                              int QueryHandle,
                              int Num,
                              out string KResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Num`	the number of parts to skip
`KResultList`	returns an XML string containing the database key and the field contents of the part defined by the `KfieldList` of the previous call to `GetFirstPartQuery`
`returns`	0 on success

XML structure of returned `KResultList`: See `GetFirstPartQuery`.

3.5.15 GetFirstCharQuery

This method retrieves information about the first characteristic the query can return for a given part. Together with `GetNextCharQuery` it can be used to create a list of characteristics of a given part.

Syntax:

```
public int GetFirstCharQuery( int Handle,
                              int QueryHandle,
                              int Part_Key,
                              string KFieldList,
                              out string KResultList);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Part_Key`	the database key of the part of which the characteristics are to be found
`KFieldList`	the string representation of an XML element which contains a list of the K-fields which have to be returned
`KResultList`	returns an XML string containing the database key and the field contents of the first characteristic defined by `KFieldList`
`returns`	0 on success

XML structure of `KFieldList` (example):

```
<FieldList>
      <Field key="2001" />
```


g-das@q-das.de • Tel. 06201/3941-0

```
      <Field key="2002" />
</FieldList>
```

XML structure of returned `KResultList` (example):

```
<KFieldList key="1">
      <Field value="1"/>
      <Field value=" Posn Jnrl P4"/>
</KFieldList>
```

3.5.16 GetNextCharQuery

After a successful call of `GetFirstCharQuery` further characteristics can be retrieved by calling this method as long as it returns 0 (zero).

Syntax:

```
public int GetNextCharQuery(  int Handle,
                              int QueryHandle,
                              out string KResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`KResultList`	returns an XML string containing the database key and the field contents of the characteristic defined by the `KFieldList` of the previous call to `GetFirstCharQuery`
`Returns`	0 on success

XML structure of returned `KResultList`: See `GetFirstCharQuery`.

3.5.17 SkipCharQuery

This method does the same as `GetNextCharQuery`, but skips a given number of characteristics before returning characteristic data.

Syntax:

```
public int SkipCharQuery(     int Handle,
                              int QueryHandle,
                              int Num,
                              out string KResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Num`	the number of parts to skip
`KResultList`	returns an XML string containing the database key and the field contents of the characteristic defined by the `KfieldList` of the previous call to `GetFirstCharQuery`
`returns`	0 on success

XML structure of returned `KResultList`: See `GetFirstCharQuery`.

q-das@q-das.de • Tel. 06201/3941-0

3.5.18 GetFirstValueQuery

This method retrieves information about the first measurement value the query can return for a given part and characteristic. Together with `GetNextValueQuery` it can be used to create a list of measurement values for a given part and characteristic.

Syntax:

```
public int GetFirstValueQuery(      int Handle,
                                    int QueryHandle,
                                    int Part_Key,
                                    int Char_Key,
                                    string KFieldList,
                                    out string KResultList);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Part_Key`	the database key of the part
`Char_Key`	the database key of the characteristic
`KFieldList`	the string representation of an XML element which contains a list of the K-fields which have to be returned (see `GetFirstPart`)
`KResultList`	returns an XML string containing the database key and the field contents of the first measurement value (see `GetFirstPart`)
`returns`	0 on success

3.5.19 GetNextValueQuery

After a successful call of `GetFirstCharQuery` further measurement values can be retrieved by calling this method as long as it returns 0 (zero).

Syntax:

```
public int GetNextValueQuery( int Handle,
                              int QueryHandle,
                              out string KResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`KResultList`	returns an XML string containing the database key and the field contents of the current measurement value (see `GetFirstPart`)
`Returns`	0 on success

3.5.20 SkipValueQuery

This method does the same as `GetNextValueQuery`, but skips a given number of measurement values before returning data.

q-das@q-das.de • Tel. 06201/3941-0

Syntax:

```
public int SkipValueQuery(    int Handle,
                              int QueryHandle,
                              int Num,
                              out string KResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Num`	the number of parts to skip
`KResultList`	returns an XML string containing the database key and the field contents of the current measurement value (see `GetFirstPart`)
`Returns`	0 on success

3.5.21 ExecuteQuery

This method loads the queried data (parts, characteristics, measurement values) into the internal structure of the qs-STAT session so that they can be evaluated and reported in a later step.

Syntax:

```
public int ExecuteQuery(      int Handle,
                              int QueryHandle,
                              string Part_Char_List );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Part_Char_List`	a string representation of an XML element containing the part and (optionally) the characteristics to be loaded (for XML structure, see `LoadQueryExt`). If `Part_Char_List` is empty, then all data the query can return are loaded.
`Returns`	0 on success

3.5.22 ExecuteQuery_Ext

Like `ExecuteQuery`, this method loads the queried data into the internal structure of the qs-STAT session. In addition to the latter it is possible to activate the write mode so that it is possible to change data in another step. It is also possible to decide if measurement values are to be loaded or not.

Syntax:

```
public int ExecuteQuery_Ext(  int Handle,
                              int QueryHandle,
                              bool ReadWrite,
                              bool LoadValues,
                              string Part_Char_List );
```


q-das@q-das.de • Tel. 06201/3941-0

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`ReadWrite`	enables change mode
`LoadValues`	enables loading of measurement values
`Part_Char_List`	a string representation of an XML element containing the part and (optionally) the characteristics to be loaded (for XML structure, see `LoadQueryExt`). If `Part_Char_List` is empty, then all data the query can return are loaded.
`Returns`	0 on success

3.5.23 FreeQuery

This method releases the query object in the qs-STAT session when it is no longer needed. It should be called because otherwise a memory leak would result.

Syntax:

```
public int FreeQuery(   int Handle,
                        int QueryHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`QueryHandle`	the query handle
`Returns`	0 on success

3.5.24 GetFirstFilterName

Together with `GetNextFilterName` this method can be used to get an enumeration of all filters stored in qs-STAT (in the configuration database) available to the current user under the current module.

Syntax:

```
public int GetFirstFilterName(      int Handle,
                                    out string FilterName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterName`	returns the name of the stored filter
`returns`	0 on success

3.5.25 GetNextFilterName

See above (`GetFirstFilterName`).

Syntax:

```
public int GetNextFilterName( int Handle,
                              out string FilterName );
```


g-das@q-das.de • Tel. 06201/3941-0

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterName`	the name of the stored filter
returns	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.5.26 GetFirstFilterNameExt

Together with `GetNextFilterNameExt` this method can be used to get an enumeration of all filters stored in qs-STAT (in the configuration database) available to the current user under the current module. Additionally these two methods return the level of data the filter can work on (part, characteristic, measurement value) and the identification key for each filter.

Syntax:

```
public int GetFirstFilterNameExt(   int Handle,
                                    out int FilterLevel,
                                    out int FilterIdent,
                                    out string FilterName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterName`	returns the name of the stored filter
`FilterLevel`	Returns the level of data on which the filter can be used: 0: Part level; 1: Characteristic level; 2: Measurement value level
`FilterIdent`	Returns an integer key which identifies the filter (can be used in LoadFilterByID
`returns`	0 on success

3.5.27 GetNextFilterNameExt

See above (`GetFirstFilterNameExt`).

Syntax:

```
public int GetNextFilterNameExt(    int Handle,
                                    out int FilterLevel,
                                    out int FilterIdent,
                                    out string FilterName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterName`	returns the name of the stored filter
`FilterLevel`	Returns the level of data on which the filter can be used: 0: Part level; 1: Characteristic level; 2: Measurement value level
`FilterIdent`	Returns an integer key which identifies the filter (can be used in LoadFilterByID
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.5.28 LoadFilter

This method loads a stored filter (specified by its name) into the qs-STAT session. The returned `FilterHandle` parameter identifies the filter in the session. The handle is needed in all commands that use the filter, especially `AddFilterToQuery`. When the filter is no longer in use it should be released by calling the `FreeFilter` method.

Syntax:

```
public int LoadFilter(  int Handle,
                        string FilterName,
                        out int FilterHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterName`	the name of the stored filter
`FilterHandle`	returns the filter handle to be used in further commands
`returns`	0 on success

3.5.29 LoadFilterByID

Like `LoadFilter` this method loads a stored filter into the qs-STAT session. Unlike the latter it uses the identification key to specify the filter.

Syntax:

```
public int LoadFilterByID(    int Handle,
                              int FilterIdent,
                              out string FilterName,
                              out int FilterHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterIdent`	the identification key of the filter
`FilterName`	returns the name of the stored filter
`FilterHandle`	returns the filter handle to be used in further commands
`returns`	0 on success

3.5.30 SaveFilter

This method can be used to save a filter in the configuration database.

Syntax:

```
public int SaveFilter(  int Handle,
                        string FilterName,
                        int FilterHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterName`	the name under which the filter is to be stored
`FilterHandle`	the filter handle
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.5.31 CreateFilter

This method is used to create a filter with a single filtering condition which is defined by a “K” field, a relational operator and a condition string. It returns a handle which is needed in commands which use the filter, especially `AddFilterToQuery` and `CreateFilterFromFilters`.

Syntax:

```
public int CreateFilter(      int Handle,
                              byte k_r_key,
                              int ausgabepunkt,
                              string Condition,
                              byte CompOp,
                              out int FilterHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`k_r_key`	must be 1 to filter by “K” fields
`ausgabepunkt`	the “K” field number
`Condition`	a string the “K” field has to match with
`CompOp`	a number which represents a comparing operator: 0: = (equals) 1: (contains a given substring) 2: > (greater than) 3: < (smaller than) 4: >= (greater than or equal) 5: >= (smaller than or equal)
`FilterHandle`	returns the handle of the filter object
`returns`	0 on success

3.5.32 CreateFilterFromSQL

Like `CreateFilter` this method creates a filter object and returns a corresponding handle, but the condition is defined by an SQL fragment, more precisely the part of a “SELECT” statement which follows the “WHERE” keyword (without “ORDER BY”, “GROUP BY”, …).

Syntax:

```
public int CreateFilterFromSQL(     int Handle,
                                    string FilterString,
                                    out int FilterHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterString`	the condition (e.g. “`TETEIL = 231`”)
`FilterHandle`	returns the handle of the filter object
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.5.33 CreateFilterFromFilters

This method can be used to combine some filters to a more complex filter structure. A list of existing filters (resp. their handles) is combined using a given logical operator and yields a new filter the handle of which is returned.

Syntax:

```
public int CreateFilterFromFilters( int Handle,
                                    byte Operator,
                                    string SourceFilters,
                                    out int FilterHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Operator`	a number which represents a logical operator: 0: AND 1: OR 2: NOT (in this case the list of source filters must only contain 1 filter) 3: NAND 4: NOR 5: XOR 6: XNOR
`SourceFilters`	the string representation of an XML element containing a list of filter handles
`FilterHandle`	returns the handle of the new filter object
`returns`	0 on success

Structure of `SourceFilters` XML element (example):

```
<FilterList>
      <Filter FilterHandle="51917424" />
      <Filter FilterHandle="51917568" />
</FilterList>"
```

3.5.34 FreeFilter

When a filter object is no longer in use it should be released with this method.

Syntax:

```
public int FreeFilter(  int Handle,
                        int FilterHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FilterHandle`	the filter handle
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.5.35 CreateDirectSQL

This method create an object which is able to send a random SQL command to the database and might receive back some data. The `ExecuteDirectSQL` method is needed to define and send the SQL command.

Syntax:

```
public int CreateDirectSQL(   int Handle,
                              out int SQLHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`SQLHandle`	returns the handle of the SQL command object
`returns`	0 on success

3.5.36 ExecuteDirectSQL

This method defines and sends the SQL command using the object handle defined with `CreateDirectSQL`.

Syntax:

```
public int ExecuteDirectSQL(  int Handle,
                              int SQLHandle,
                              string SQLString,
                              out string FieldList);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`SQLHandle`	the handle of the SQL command object
`SQLString`	an SQL statement which is valid for the database currently in use
`FieldList`	returns a string representation of an XML element which contains the SQL command itself as well as a list of the columns it might return
`Returns`	0 on success

Structure of `FieldList` *XML element (example):*

```
<SQL sqlString="SELECT TETEIL, TETEILNR FROM TEIL">
      <Column colNr="1">TETEIL</Column>
      <Column colNr="2">TETEILNR</Column>
</SQL>
```


q-das@q-das.de • Tel. 06201/3941-0

3.5.37 GetFirstDirectSQLRow

If `ExecuteDirectSQL` has been called before with an SQL statement that can return data, this method and in case of success `GetNextDirectSQLRow` can be called to collect all the results the SQL statement yields.

Syntax:

```
public int GetFirstDirectSQLRow(    int Handle,
                                    int SQLHandle,
                                    bool pdf,
                                    out string ResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`SQLHandle`	the handle of the SQL command object
`pdf`	for internal use only, should be set to “false”
`ResultList`	returns a string representation of an XML element which contains a list of the contents of each returned column
`Returns`	0 on success

Structure of `ResultList` *XML element (example):*

```
<SQL>
      <Column colNr="1">1</Column>
      <Column colNr="2">P1</Column>
</SQL>
```

3.5.38 GetNextDirectSQLRow

If `ExecuteDirectSQL` has been called before with an SQL statement that can return data, this method and in case of success `GetNextDirectSQLRow` can be called to collect all the results the SQL statement yields.

Syntax:

```
public int GetFirstDirectSQLRow(    int Handle,
                                    int SQLHandle,
                                    bool pdf,
                                    out string ResultList );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`SQLHandle`	the handle of the SQL command object
`pdf`	for internal use only, should be set to “false”
`ResultList`	returns a string representation of an XML element which contains a list of the contents of each returned column
`Returns`	0 on success

Structure of `ResultList` *XML element (example):*

```
<SQL>
      <Column colNr="1">2</Column>
      <Column colNr="2">P2</Column>
</SQL>
```


q-das@q-das.de • Tel. 06201/3941-0

3.5.39 FreeDirectSQL

This method should be called to avoid memory leaks when the SQL command object is no longer in use.

Syntax:

```
public int FreeDirectSQL(     int Handle,
                              int SQLHandle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`SQLHandle`	the handle of the SQL command object
`Returns`	0 on success

3.5.40 RecentSerNo_First

This method yields information about the most recently measured serial number for a given part.

Syntax:

```
public int RecentSerNo_First( int Handle,
                              int PartKey,
                              out string SerialNo);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`PartKey`	the database key of the part to find the most recent serial number for
`SerialNo`	returns the most recent serial number
`Returns`	0 on success

3.5.41 RecentSerNo_Next

This method returns the serial number which precedes the number of the last call of `RecentSerNo_First` or `RecentSerNo_Next` .

Syntax:

```
public int RecentSerNo_Next(  int Handle,
                              out string SerialNo );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`SerialNo`	returns the serial number preceding the serial number of the last call
`Returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.5.42 SaveToDB

This method saves the data of the current qs-STAT session into the database (always as a new part).

Syntax:
```
public int SaveToDB(    int Handle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Returns`	0 on success

3.5.43 SaveChangesToDB

This method saves the data of the current qs-STAT session into the database (changing the existing part and its characteristics).

Syntax:
```
public int SaveChangesToDB(   int Handle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Returns`	0 on success

3.6 Evaluation and Data Handling Commands

3.6.1 EvaluateAllChars

This method evaluates the data of the current qs-STAT session.

Syntax:
```
public int EvaluateAllChars(  int Handle );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Returns`	0 on success

3.6.2 EvaluateChar

This method evaluates one characteristic in the current qs-STAT session.

Syntax:
```
public int EvaluateChar(      int Handle,
                              int Part_Nr,
                              int Char_Nr,
```


q-das@q-das.de • Tel. 06201/3941-0

```
                    bool SPCEvaluation,
                    int Reserve1,
                    int Reserve2 );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Part_Nr`	the sequential number of the part
`Char_Nr`	the sequential number of the characteristic to evaluate
`SPCEvaluation`	?
`Reserve1`	not used
`Reserve2`	not used
`Returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.6.3 GetGlobalInfo

Depending on `Op_Code` this method returns information about the number of loaded parts, characteristics etc.

Syntax:

```
public int GetGlobalInfo(     int Handle,
                              int Part_Nr,
                              int Char_Nr,
                              int Op_Code,
                              out int ret );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Part_Nr`	the sequential number of the part to get information about or 0
`Char_Nr`	the sequential number of the characteristic to get information about or 0
`Op_Code`	the code which defines the kind of information needed (see table below)
`ret`	the returned value
`Returns`	0 on success

Op_Codes:

1	number of loaded parts
2	total number of loaded selected characteristics
3	number of loaded selected characteristics for a given part (defined by `Part_Nr`)
4	number of measurement values for a given characteristic (defined by `Part_Nr` and `Char_Nr`)
5	number of loaded selected and deselected characteristics for a given part (defined by `Part_Nr`)
6	total number of loaded selected and deselected characteristics
10	gets the index number of the first evaluated value
11	gets the index number of the last evaluated value
12	gets the index number of the first evaluated subgroup
13	gets the index number of the last evaluated subgroup
101	size of FIFO buffer
111	number of evaluated subgroups in procella
200	returns the current module key
201	returns the allowed number of concurrent web service accesses (licenses)
202	returns the remaining number of days on a time-limited qs-STAT license
300	returns the key of the user in the current qs-STAT session
301	returns the key of the current user's user group

g-das@q-das.de • Tel. 06201/3941-0

3.6.4 GetPartInfo

This method returns the contents of a specified field of a part.

Syntax:

```
public int GetPartInfo( int Handle,
                        int FieldNr,
                        int Part_Nr,
                        int Char_Nr,
                        out string KfieldValue );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FieldNr`	the key number of a “K” field the contents of which is to be returned
`Part_Nr`	the sequential number of the part
`Char_Nr`	not used, should be 0
`KfieldValue`	returns the contents of the “K” field
`Returns`	0 on success

3.6.5 GetCharInfo

This method returns the contents of a specified field of a characteristic.

Syntax:

```
public int GetCharInfo( int Handle,
                        int FieldNr,
                        int Part_Nr,
                        int Char_Nr,
                        out string KfieldValue );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FieldNr`	the key number of a “K” field the contents of which is to be returned
`Part_Nr`	the sequential number of the part
`Char_Nr`	the sequential number of the characteristic
`KfieldValue`	returns the contents of the “K” field
`Returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.6.6 GetValueInfo

This method returns the contents of a specified field of a measurement value.

Syntax:

```
public int GetValueInfo(      int Handle,
                              int FieldNr,
                              int Part_Nr,
                              int Char_Nr,
                              int Value_Nr,
                              byte bTransformation,
                              out string KfieldValue );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`FieldNr`	the key number of a “K” field the contents of which is to be returned
`Part_Nr`	the sequential number of the part
`Char_Nr`	the sequential number of the characteristic
`Value_Nr`	the sequential number of the measurement value
`bTransformation`	0 – not transformed; 1 – transformed; 2 – re-transformed
`KfieldValue`	returns the contents of the “K” field
`Returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.6.7 GetSingleValueEx

This method returns more specified information about a measurement value.

Syntax:

```
public int GetSingleValueEx(  int Handle,
                              int Part_Nr,
                              int Char_Nr,
                              int ResultKey,
                              int ResultSubKey,
                              int reserved1,
                              int reserved2,
                              int OutputType,
                              bool Transformation,
                              bool Sorted,
                              int Value_sp_nr,
                              int gc_n,
                              int gc_r,
                              int gc_k,
                              int gc_l,
                              out string Result_str,
                              out double Result_dbl );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Part_Nr`	the sequential number of the part
`Char_Nr`	the sequential number of the characteristic
`ResultKey`	the key number of an “V” field the contents of which is to be returned
`ResultSubKey`	the subkey number of the “V” field
`reserved1`	not used
`reserved2`	not used
`OutputType`	describes what kind of output is needed (see table below)
`Transformation`	
`Sorted`	
`Value_sp_nr`	the sequential number of the measurement value or sample
`gc_n`	
`gc_r`	
`gc_k`	
`gc_l`	
`Result_str`	returns the result as a string
`Result_dbl`	returns the result as a float number (if possible)
`Returns`	0 on success

Output type:

Value	Name of constant	Description
1	AA_LONGTEXT_C	a description of the field is returned (long form)
2	AA_SHORTTEXT_C	a description of the field is returned (short form)
4	AA_CONTENT_C	the field value(s) is/are returned
512	AA_CONTTEXT_C	a text which describes the field value(s) is returned

q-das@q-das.de • Tel. 06201/3941-0

3.6.8 GetStatResult

After evaluation this method returns the contents of a specified result field (“R” field).

Syntax:

```
public int GetStatResult(     int Handle,
                              int ResultKey,
                              int Part_Nr,
                              int Char_Nr,
                              byte bTransformation,
                              out string StatResult_str,
                              out double StatResult_dbl);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`ResultKey`	the key number of an “R” field the contents of which is to be returned
`Part_Nr`	the sequential number of the part
`Char_Nr`	the sequential number of the characteristic
`bTransformation`	0 – not transformed; 1 – transformed; 2 – re-transformed
`Result_str`	returns the result as a string
`Result_dbl`	returns the result as a float number (if possible)
`Returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.6.9 GetStatResultEx

After evaluation this method returns the contents of a specified result field (“R” field) in a more complex way than `GetStatResult`.

```
public int GetStatResultEx(   int Handle,
                              int ResultKey,
                              int ResultSubKey,
                              int Part_Nr,
                              int Char_Nr,
                              int Group_Nr,
                              int OutputType,
                              int TransformationType,
                              int reserve1,
                              int reserve2,
                              out string StatResult_str1,
                              out string StatResult_str2,
                              out string StatResult_str3,
                              out string StatResult_str4,
                              out string StatResult_str5,
                              out int OutputCount,
                              out double StatResult_dbl1,
                              out double StatResult_dbl2,
                              out double StatResult_dbl3,
                              out double StatResult_dbl4,
                              out double StatResult_dbl5);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`ResultKey`	the key number of an “R” field the contents of which is to be returned
`ResultSubKey`	the subkey number of the “R” field
`Part_Nr`	the sequential number of the part
`Char_Nr`	the sequential number of the characteristic
`Group_Nr`	the sequential number of the group
`OutputType`	describes what kind of output is needed (see table in chapter 3.6.7)
`TransformationType`	0 – not transformed; 1 – transformed; 2 – re-transformed
`reserve1`	not used
`Reserve2`	not used
`StatResult_str1`	returns the first part of a compound result as a string
`StatResult_str2`	returns the second part of a compound result as a string
`StatResult_str3`	returns the third part of a compound result as a string
`StatResult_str4`	returns the fourth part of a compound result as a string
`StatResult_str5`	returns the fifth part of a compound result as a string
`OutputCount`	returns the number of components of a compound result
`StatResult_dbl1`	returns the first part of a compound result as a float
`StatResult_dbl2`	returns the second part of a compound result as a float
`StatResult_dbl3`	returns the third part of a compound result as a float
`StatResult_dbl4`	returns the fourth part of a compound result as a float
`StatResult_dbl5`	returns the fifth part of a compound result as a float
`Returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.6.10 SetKey

This method allows to change K-field data.

```
public int SetKey (     int Handle,
                        int PartNr,
                        int CharNr,
                        int GroupNr_Nr,
                        int ValueNr,
                        int Key,
                        int SubKey,
                        int Data,
                        int CharRange);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`PartNr`	the sequential number of the part
`CharNr`	the sequential number of the characteristic
`GroupNr_Nr`	the sequential number of the group
`ValueNr`	the sequential number of the measurement value
`Key`	the K-field to be changed
`SubKey`	the subkey (usually 0)
`Data`	the new field value
`CharRange`	describes which characteristics are affected by the changing: 1: All characteristics of all parts 2: All characteristics of the part defined by `PartNr`. 16: The characteristic defined by `PartNr` and `CharNr`.
`Returns`	0 on success

3.6.11 GetFirstChildNode

This method returns the first child node of a given node in the parts/characterisics tree. Both nodes are identified by their combination of part number, group number and characteristic number. Together with `GetNextSiblingNode` this method allows to navigate through parent/child structures of the parts/characterisics tree.

```
public int GetFirstChildNode (      int Handle,
                                    int PartNo_Parent,
                                    int GroupNo_Parent,
                                    int CharNo_Parent,
                                    out int PartNo,
                                    out int GroupNo,
                                    out int CharNo);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`PartNo_Parent`	the sequential part number of the parent node
`GroupNo_Parent`	the sequential group number of the parent node
`CharNo_Parent`	the sequential characteristic number of the parent node
`PartNo`	returns the sequential part number of the child node if available
`GroupNo`	returns the sequential group number of the child node if available

q-das@q-das.de • Tel. 06201/3941-0

CharNo	returns the sequential charact. number of the child node if available
Returns	0 on success, nonzero if no child available

3.6.12 GetNextSiblingNode

This method returns the next node having the same parent as a given node in the parts/characterisics tree. Both nodes are identified by their combination of part number, group number and characteristic number. Together with `GetFirstChildNode` this method allows to navigate through parent/child structures of the parts/characterisics tree.

```
public int GetNextSiblingNode (     int Handle,
                                    int PartNo_Current,
                                    int GroupNo_Current,
                                    int CharNo_Current,
                                    out int PartNo,
                                    out int GroupNo,
                                    out int CharNo);
```

Parameters:

Handle	the handle of the qs-STAT session
PartNo_Current	the sequential part number of the given node
GroupNo_Current	the sequential group number of the given node
CharNo_Current	the sequential characteristic number of the given node
PartNo	returns the sequential part number of the sibling node if available
GroupNo	returns the sequential group number of the sibling node if available
CharNo	returns the sequential charact. number of the sibling node if available
Returns	0 on success, nonzero if no sibling node available

3.7 Commands to handle Graphics

3.7.1 GetFirstGraphic

This method returns the first qs-STAT graphic available in the given module. Together with `GetNextGraphic` this method can be used to build a list of qs-STAT graphics.

Syntax:

```
public int GetFirstGraphic(   int Handle,
                              int Module,
                              out int GraphicNr,
                              out string GraphicName );
```

Parameters:

Handle	the handle of the qs-STAT session
Module	the module key
GraphicNr	returns the graphic key number

q-das@q-das.de • Tel. 06201/3941-0

`GraphicName`	returns the name of the graphic
`Returns`	0 on success

3.7.2 GetNextGraphic

This method returns another qs-STAT graphic available in the given module. After a successful call of `GetFirstGraphic` this method can be called as long as it returns 0 to build a list of qs-STAT graphics.

Syntax:

```
public int GetNextGraphic(    int Handle,
                              int Module,
                              out int GraphicNr,
                              out string GraphicName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Module`	the module key
`GraphicNr`	returns the graphic key number
`GraphicName`	returns the name of the graphic
`Returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.7.3 SkipGraphic

This method can be used like `GetNextGraphic`, but skips a given number of graphics before returning a result.

Syntax:

```
public int SkipGraphic( int Handle,
                        int Module,
                        int Num,
                        out int GraphicNr,
                        out string GraphicName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Module`	the module key
`Num`	the number of graphics to skip
`GraphicNr`	returns the graphic key number
`GraphicName`	returns the name of the graphic
`Returns`	0 on success

3.7.4 GetGraphicName

This method returns the name of a graphic with a given key number in a given module.

Syntax:

```
public int GetGraphicName(    int Handle,
                              int Module,
                              int GraphicNr,
                              out string GraphicName);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Module`	the module key
`GraphicNr`	the graphic key number
`GraphicName`	returns the name of the graphic
`Returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.7.5 GetGraphicNameExt

This method returns the long and short names of a graphic defined by its key, subkey and additional key numbers.

Syntax:
```
public int GetGraphicNameExt( int Handle,
                              int GraphicKey,
                              int GraphicSubKey,
                              int GraphicAddKey,
                              out string GraphicLongName,
                              out string GraphicShortName);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicKey`	the graphic key number
`GraphicSubKey`	the graphic subkey number
`GraphicAddKey`	the additional graphic key number (usually 0)
`GraphicLongName`	returns the long name of the graphic
`GraphicShortName`	returns the short name of the graphic
`Returns`	0 on success

3.7.6 GetGraphic

This method creates and returns a qs-STAT graphic (as a bitmap) for a given part and characteristic with a given size, using the data of the current qs-STAT session.

Syntax:
```
public int GetGraphic(  int Handle,
                        int Graphic_Nr,
                        int Part_Nr,
                        int Char_Nr,
                        int Width,
                        int Height,
                        out string GraphicStr);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Graphic_Nr`	the graphic key number
`Part_Nr`	the index number of the part
`Char_Nr`	the index number of the characteristic
`Width`	the desired width of the graphic
`Height`	the desired height of the graphic
`GraphicStr`	returns the string representation of an XML element containing the graphic (see description below)
`Returns`	0 on success

Structure of the XML element:
```
<Test>
      <Image format="bmp">Binary data transformed to 64-bit-chars</Image>
</Test>
```


q-das@q-das.de • Tel. 06201/3941-0

To get the graphic, the inner text of the `Image` XML element must be converted back to binary data, e.g. by using the `Convert.FromBase64String` method (C#).

3.7.7 GetGraphicExt

This method is an extended version of `GetGraphic`. It allows a more detailed descriptin of the desired graphic, and the file format of the graphic may be selected.

Syntax:

```
public int GetGraphicExt(     int Handle,
                              int GraphicNr,
                              int GraphicSubNr,
                              int PartNr_x,
                              int CharNr_x,
                              int ValueNr_x,
                              int NrOfValues_x,
                              int GroupNr_x,
                              int PartNr_y,
                              int CharNr_y,
                              int ValueNr_y,
                              int NrOfValues_y,
                              int GroupNr_y,
                              int Width,
                              int Height,
                              int NrOfColumns,
                              int NrOfRows,
                              int ConfigID,
                              int OutputFormat,
                              out string GraphicStr);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicNr`	the graphic key number
`GraphicSubNr`	the graphic subkey
`PartNr_x`	the index number of the part
`CharNr_x`	the index number of the characteristic
`ValueNr_x`	the index number of the first value to show
`NrOfValues_x`	the number of values to show
`GroupNr_x`	the index number of the group
`PartNr_y`	the index number of the second part (in graphics which require two characteristics)
`CharNr_y`	the index number of the second characteristic
`ValueNr_y`	the index number of the first value to show in the second characteristic
`NrOfValues_y`	the number of values to show in the second characteristic
`GroupNr_y`	the index number of the second group
`Width`	the desired width of the graphic
`Height`	the desired height of the graphic
`NrOfColumns`	the number of columns in a tabular graphic
`NrOfRows`	the number of rows in a tabular graphic
`ConfigID`	0 or the key number of another graphic where the desired graphic can be

q-das@q-das.de • Tel. 06201/3941-0

	embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
`OutputFormat`	0 – Bitmap; 1 – JPEG; 2 – Enhanced Metafile; 3 – Metafile
`GraphicStr`	returns the string representation of an XML element containing the graphic (see description in `GetGraphic`)
`Returns`	0 on success

3.7.8 GetGraphicExt2

This method is the second extended version of `GetGraphic`. In addition to `GetGraphicExt` it allows to mark fields in certain graphics, e.g. alarm tables. It is also possible to receive the follow-ups of tabular graphics (lists) when the information does not fit in a single graphic page.

Syntax:

```
public int GetGraphicExt2(    int Handle,
                              int GraphicNr,
                              int GraphicSubNr,
                              int PartNr_x,
                              int GroupNr_x,
                              int CharNr_x,
                              int ValueNr_x,
                              int NrOfValues_x,
                              int PartNr_y,
                              int GroupNr_y,
                              int CharNr_y,
                              int ValueNr_y,
                              int NrOfValues_y,
                              int PartNr_MarkStart,
                              int GroupNr_MarkStart,
                              int CharNr_MarkStart,
                              int ValueNr_MarkStart,
                              int PartNr_MarkEnd,
                              int GroupNr_MarkEnd,
                              int CharNr_MarkEnd,
                              int ValueNr_MarkEnd,
                              int NrOfColumns,
                              int NrOfRows,
                              int Width,
                              int Height,
                              int ConfigID,
                              int PageBlockStart_X,
                              int PageBlockStart_Y,
                              int OutputFormat,
                              out int NrOfBlocksInPage_X,
                              out int NrOfBlocksInPage_Y,
                              out string GraphicStr);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicNr`	the graphic key number
`GraphicSubNr`	the graphic subkey
`PartNr_x`	the index number of the part
`GroupNr_x`	the index number of the group

q-das@q-das.de • Tel. 06201/3941-0

CharNr_x	the index number of the characteristic
ValueNr_x	the index number of the first value to show
NrOfValues_x	the number of values to show
PartNr_y	the index number of the second part (in graphics which require two characteristics)
GroupNr_y	the index number of the second group
CharNr_y	the index number of the second characteristic
ValueNr_y	the index number of the first value to show in the second characteristic
NrOfValues_y	the number of values to show in the second characteristic
PartNr_MarkStart	the index number of the part where the marking begins
GroupNr_MarkStart	the index number of the group where the marking begins
CharNr_MarkStart	the index number of the characteristic where the marking begins
ValueNr_MarkStart	the index number of the value where the marking begins
PartNr_MarkEnd	the index number of the part where the marking ends
GroupNr_MarkEnd	the index number of the group where the marking ends
CharNr_MarkEnd	the index number of the characteristic where the marking ends
ValueNr_MarkEnd	the index number of the value where the marking ends
NrOfColumns	the number of columns in a tabular graphic
NrOfRows	the number of rows in a tabular graphic
Width	the desired width of the graphic
Height	the desired height of the graphic
ConfigID	0 or the key number of another graphic where the desired graphic can be embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
PageBlockStart_X	in a tabular graphic the item number where the graphic starts (first dimension)
PageBlockStart_Y	in a tabular graphic the item number where the graphic starts (second dimension)
OutputFormat	0 – Bitmap; 1 – JPEG; 2 – Enhanced Metafile; 3 – Metafile
NrOfBlocksInPage_X	returns (in a tabular graphic) the number of blocks/entries created in the first dimension so that a follow-up graphic can be created by adding this value to the previous value of PageBlockStart_X
NrOfBlocksInPage_Y	returns (in a tabular graphic) the number of blocks/entries created in the second dimension
GraphicStr	returns the string representation of an XML element containing the graphic (see description in GetGraphic)
Returns	0 on success

3.7.9 GetGraphicExt3

This method is the third extended version of GetGraphic. In addition to GetGraphicExt2 it allows to define multiple characteristics, e.g. to create overlaid value charts or overview graphics with a reduced list of characteristics. To do so, the int parameters from PartNr_x to NrOfValues_y have been replaced by string parameters, which allows to pass multiple pairs of part number / characteristic number to the function as comma-separated lists, e.g. “1,1,1,1” as PartNr_x and “1,2,3,4” as CharNr_x to display characteristics 1 to 4 of part 1.

Syntax:

```
public int GetGraphicExt2(    int Handle,
```


q-das@q-das.de • Tel. 06201/3941-0

```
int GraphicNr,
int GraphicSubNr,
string PartNr_x,
string GroupNr_x,
string CharNr_x,
string ValueNr_x,
string NrOfValues_x,
string PartNr_y,
string GroupNr_y,
string CharNr_y,
string ValueNr_y,
string NrOfValues_y,
int Handle_Mark,
int PartNr_MarkStart,
int GroupNr_MarkStart,
int CharNr_MarkStart,
int ValueNr_MarkStart,
int PartNr_MarkEnd,
int GroupNr_MarkEnd,
int CharNr_MarkEnd,
int ValueNr_MarkEnd,
int NrOfColumns,
int NrOfRows,
int Width,
int Height,
int ConfigID,
int PageBlockStart_X,
int PageBlockStart_Y,
int OutputFormat,
out int NrOfBlocksInPage_X,
out int NrOfBlocksInPage_Y,
out string GraphicStr);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicNr`	the graphic key number
`GraphicSubNr`	the graphic subkey
`PartNr_x`	a comma-separated list of part index numbers (must have the same number of entries as `CharNr_x` because part numbers and chaacteristic numbers must correspond).
`GroupNr_x`	the index number of the group
`CharNr_x`	a comma-separated list of characteristic index numbers
`ValueNr_x`	the index number of the first value to show
`NrOfValues_x`	the number of values to show
`PartNr_y`	the index number of the second part (in graphics which require two characteristics)
`GroupNr_y`	the index number of the second group
`CharNr_y`	the index number of the second characteristic
`ValueNr_y`	the index number of the first value to show in the second characteristic
`NrOfValues_y`	the number of values to show in the second characteristic
`Handle_Mark`	(not yet used)
`PartNr_MarkStart`	the index number of the part where the marking begins
`GroupNr_MarkStart`	the index number of the group where the marking begins
`CharNr_MarkStart`	the index number of the characteristic where the marking begins

q-das@q-das.de • Tel. 06201/3941-0

`ValueNr_MarkStart`	the index number of the value where the marking begins
`PartNr_MarkEnd`	the index number of the part where the marking ends
`GroupNr_MarkEnd`	the index number of the group where the marking ends
`CharNr_MarkEnd`	the index number of the characteristic where the marking ends
`ValueNr_MarkEnd`	the index number of the value where the marking ends
`NrOfColumns`	the number of columns in a tabular graphic
`NrOfRows`	the number of rows in a tabular graphic
`Width`	the desired width of the graphic
`Height`	the desired height of the graphic
`ConfigID`	0 or the key number of another graphic where the desired graphic can be embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
`PageBlockStart_X`	in a tabular graphic the item number where the graphic starts (first dimension)
`PageBlockStart_Y`	in a tabular graphic the item number where the graphic starts (second dimension)
`OutputFormat`	0 – Bitmap; 1 – JPEG; 2 – Enhanced Metafile; 3 – Metafile
`NrOfBlocksInPage_X`	returns (in a tabular graphic) the number of blocks/entries created in the first dimension so that a follow-up graphic can be created by adding this value to the previous value of `PageBlockStart_X`
`NrOfBlocksInPage_Y`	returns (in a tabular graphic) the number of blocks/entries created in the second dimension
`GraphicStr`	returns the string representation of an XML element containing the graphic (see description in `GetGraphic`)
`Returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.7.10 GetDataPositionByCoordExt

This method can be used to identify the data showed in a graphic on a given position, e.g. when the user clicks on a graphic.

Syntax:

```
public int GetDataPositionByCoordExt(     int Handle,
                                          int GraficNr,
                                          int GraficSubNr,
                                          int PartNr_x,
                                          int CharNr_x,
                                          int ValueNr_x,
                                          int NrOfValues_x,
                                          int GroupNr_x,
                                          int PartNr_y,
                                          int CharNr_y,
                                          int ValueNr_y,
                                          int NrOfValues_y,
                                          int GroupNr_y,
                                          int NrOfColumns,
                                          int NrOfRows,
                                          byte GetPositionOnly,
                                          int Width, int Height,
                                          int ConfigID,
                                          int OutputFormat,
                                          ref int X,
                                          ref int Y,
                                          out int ChildHandle,
                                          out int PartNrOut_x,
                                          out int GroupNrOut_x,
                                          out int CharNrOut_x,
                                          out int ValueNrOut_x,
                                          out int PartNrOut_y,
                                          out int GroupNrOut_y,
                                          out int CharNrOut_y,
                                          out int ValueNrOut_y,
                                          out string GraphicStr);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicNr`	the graphic key number
`GraphicSubNr`	the graphic subkey
`PartNr_x`	the index number of the part
`CharNr_x`	the index number of the characteristic
`ValueNr_x`	the index number of the first value to show
`NrOfValues_x`	the number of values to show
`GroupNr_x`	the index number of the group
`PartNr_y`	the index number of the second part (in graphics which require two characteristics)
`CharNr_y`	the index number of the second characteristic
`ValueNr_y`	the index number of the first value to show in the second characteristic
`NrOfValues_y`	the number of values to show in the second characteristic
`GroupNr_y`	the index number of the second group
`NrOfColumns`	the number of columns in a tabular graphic
`NrOfRows`	the number of rows in a tabular graphic

q-das@q-das.de • Tel. 06201/3941-0

`GetPositionOnly`	0 – only drawing; 1 – position detection only; 2 – drawing and position detection
`Width`	the desired width of the graphic
`Height`	the desired height of the graphic
`ConfigID`	0 or the key number of another graphic where the desired graphic can be embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
`OutputFormat`	0 – Bitmap; 1 – JPEG; 2 – Enhanced Metafile; 3 – Metafile
`X`	the pixel position (from left) of the point to get data from
`Y`	the pixel position (from top) of the point to get data from
`ChildHandle`	returns the user key
`PartNrOut_x`	returns the index number of the part found
`GroupNrOut_x`	returns the index number of the group found
`CharNrOut_x`	returns the index number of the characteristic found
`ValueNrOut_x`	returns the index number of the measurement value found
`PartNrOut_y`	returns the index number of the second part found (in graphics which require two characteristics)
`GroupNrOut_y`	returns the index number of the second group found
`CharNrOut_y`	returns the index number of the second characteristic found
`ValueNrOut_y`	returns the index number of the second value found
`GraphicStr`	returns the string representation of an XML element containing the graphic (see description in `GetGraphic`)
`Returns`	the index number of the second group

g-das@q-das.de • Tel. 06201/3941-0

3.7.11 GetDataPositionByCoordExt3

This method can be used to identify the data showed in a graphic on a given position, e.g. when the user clicks on a graphic.

Syntax:

```
public int GetDataPositionByCoordExt3(    int Handle,
                                          int GraficNr,
                                          int GraficSubNr,
                                          int PartNr_x,
                                          int CharNr_x,
                                          int ValueNr_x,
                                          int NrOfValues_x,
                                          int GroupNr_x,
                                          int PartNr_y,
                                          int CharNr_y,
                                          int ValueNr_y,
                                          int NrOfValues_y,
                                          int GroupNr_y,
                                          int NrOfColumns,
                                          int NrOfRows,
                                          byte GetPositionOnly,
                                          int Width,
                                          int Height,
                                          int ConfigID,
                                          int OutputFormat,
                                          int PageBlockStartX,
                                          int PageBlockStartY,
                                          ref int X,
                                          ref int Y,
                                          out int ChildHandle,
                                          out int PartNrOut_x,
                                          out int GroupNrOut_x,
                                          out int CharNrOut_x,
                                          out int ValueNrOut_x,
                                          out int ValueNrArt_out_x,
                                          out int AusgabepunktFeldArtOut,
                                          out Int64 AusgabepunktFeldOut,
                                          out int AusgabepunktOut,
                                          out int AusgabepunktSubkeyOut,
                                          out int PartNrOut_y,
                                          out int GroupNrOut_y,
                                          out int CharNrOut_y,
                                          out int ValueNrOut_y,
                                          out int NrOfPageBlocksX,
                                          out int NrOfPageBlocksY,
                                          out string XmlInfoStr,
                                          out string GraphicStr);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicNr`	the graphic key number
`GraphicSubNr`	the graphic subkey
`PartNr_x`	the index number of the part
`CharNr_x`	the index number of the characteristic
`ValueNr_x`	the index number of the first value to show

q-das@q-das.de • Tel. 06201/3941-0

NrOfValues_x	the number of values to show
GroupNr_x	the index number of the group
PartNr_y	the index number of the second part (in graphics which require two characteristics)
CharNr_y	the index number of the second characteristic
ValueNr_y	the index number of the first value to show in the second characteristic
NrOfValues_y	the number of values to show in the second characteristic
GroupNr_y	the index number of the second group
NrOfColumns	the number of columns in a tabular graphic
NrOfRows	the number of rows in a tabular graphic
GetPositionOnly	0 – only drawing; 1 – position detection only; 2 – drawing and position detection
Width	the desired width of the graphic
Height	the desired height of the graphic
ConfigID	0 or the key number of another graphic where the desired graphic can be embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
OutputFormat	0 – Bitmap; 1 – JPEG; 2 – Enhanced Metafile; 3 – Metafile
PageBlockStartX	If the graphic is an overview graphic, it might need more than one page. In this case these parameters define starting values of a page; in other cases they should be set to 0. See also NrOfPageBlocksX, NrOfPageBlocksY.
PageBlockStartY	
X	the pixel position (from left) of the point to get data from
Y	the pixel position (from top) of the point to get data from
ChildHandle	returns the user key
PartNrOut_x	returns the index number of the part found
GroupNrOut_x	returns the index number of the group found
CharNrOut_x	returns the index number of the characteristic found
ValueNrOut_x	returns the index number of the measurement value found
ValueNrArt_out_x	
AusgabepunktFeldArtOut	returns the kind of output field found (e.g. K-field, R-field)
AusgabepunktFeldOut	returns the output mode of the field (e.g. contents, long field name, short field name etc.
AusgabepunktOut	returns the output key number
AusgabepunktSukeyOut	returns the output subkey number
PartNrOut_y	returns the index number of the second part found (in graphics which require two characteristics)
GroupNrOut_y	returns the index number of the second group found
CharNrOut_y	returns the index number of the second characteristic found
ValueNrOut_y	returns the index number of the second value found
NrOfPageBlocksX	returns the number of entries in one page in x/y direction. To display the next page, these values have to be added to PageBlockStartX and PageBlockStartY.
NrOfPageBlocksY	
XmlInfoStr	is intended to contain further information in XML format (not used until now)
GraphicStr	returns the string representation of an XML element containing the graphic (see description in GetGraphic)
Returns	the index number of the second group

q-das@q-das.de • Tel. 06201/3941-0

3.7.12 GetGraphicPages

This method retrieves information about how many graphic pages could be produced and how many blocks/entries they would contain when creating a tabular graphic with `GetGraphicExt2`.

Syntax:

```
public int GetGraphicPages(   int Handle,
                              int GraphicNr,
                              int GraphicSubNr,
                              int PartNr_x,
                              int GroupNr_x,
                              int CharNr_x,
                              int ValueNr_x,
                              int NrOfValues_x,
                              int PartNr_y,
                              int GroupNr_y,
                              int CharNr_y,
                              int ValueNr_y,
                              int NrOfValues_y,
                              int NrOfColumns,
                              int NrOfRows,
                              int Width,
                              int Height,
                              int ConfigID,
                              out int NrOfPages,
                              out int BlocksPerPage_X,
                              out int BlocksPerPage_Y );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicNr`	the graphic key number
`GraphicSubNr`	the graphic subkey
`PartNr_x`	the index number of the part
`GroupNr_x`	the index number of the group
`CharNr_x`	the index number of the characteristic
`ValueNr_x`	the index number of the first value to show
`NrOfValues_x`	the number of values to show
`PartNr_y`	the index number of the second part (in graphics which require two characteristics)
`GroupNr_y`	the index number of the second group
`CharNr_y`	the index number of the second characteristic
`ValueNr_y`	the index number of the first value to show in the second characteristic
`NrOfValues_y`	the number of values to show in the second characteristic
`NrOfColumns`	the number of columns in a tabular graphic
`NrOfRows`	the number of rows in a tabular graphic
`Width`	the desired width of the graphic
`Height`	the desired height of the graphic
`ConfigID`	0 or the key number of another graphic where the desired graphic

g-das@q-das.de • Tel. 06201/3941-0

	can be embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
`NrOfPages`	returns the number of consecutive pages that could be created
`BlocksPerPage_X`	returns the maximal number of blocks/entries created in the first dimension
`BlocksPerPage_Y`	returns the maximal number of blocks/entries created in the second dimension
`Returns`	0 on success

3.7.13 GetGraphArray

This method retrieves the contents of a table-like graphic as an XML structure.

Syntax:

```
public int GetGraphicPages(   int Handle,
                              int GraphicNr,
                              int GraphicSubNr,
                              int PartNr_x,
                              int GroupNr_x,
                              int CharNr_x,
                              int ValueNr_x,
                              int NrOfValues_x,
                              int PartNr_y,
                              int GroupNr_y,
                              int CharNr_y,
                              int ValueNr_y,
                              int NrOfValues_y,
                              int NrOfColumns,
                              int NrOfRows,
                              int SingleImageWidth,
                              int SingleImageHeight,
                              int ConfigID,
                              int PageBlockStart_X,
                              int PageBlockStart_Y,
                              int OutputFormat,
                              out int NrOfPages,
                              out int NrOfBlocksInPage_X,
                              out int NrOfBlocksInPage_Y,
                              out string GraphicArrayXml );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GraphicNr`	the graphic key number
`GraphicSubNr`	the graphic subkey
`PartNr_x`	the index number of the part
`GroupNr_x`	the index number of the group
`CharNr_x`	the index number of the characteristic
`ValueNr_x`	the index number of the first value to show
`NrOfValues_x`	the number of values to show
`PartNr_y`	the index number of the second part (in graphics which require two characteristics)

q-das@q-das.de • Tel. 06201/3941-0

GroupNr_y	the index number of the second group
CharNr_y	the index number of the second characteristic
ValueNr_y	the index number of the first value to show in the second characteristic
NrOfValues_y	the number of values to show in the second characteristic
NrOfColumns	the number of columns in a tabular graphic
NrOfRows	the number of rows in a tabular graphic
SingleImageWidth SingleImageHeight	In case of embedded image-like graphics these fields contain their width and height.
ConfigID	0 or the key number of another graphic where the desired graphic can be embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
PageBlockStart_X PageBlockStart_Y	Tabular graphics might need more than one page. In this case these parameters define starting values of a page; in other cases they should be set to 0. See also NrOfBlocksInPage_X, NrOfBlocksInPage_Y.
OutputFormat	0 – Bitmap; 1 – JPEG; 2 – Enhanced Metafile; 3 – Metafile
NrOfBlocksInPage_X	returns the maximal number of blocks/entries created in the first dimension
NrOfBlocksInPage_Y	returns the maximal number of blocks/entries created in the second dimension
GraphicArrayXml	Returns the graphics contents as an XML structure
Returns	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.8 Commands to handle Reports

3.8.1 GetFirstReport

Together with `GetNextReport` this method can be used to get an enumeration of all reports available in the current qs-STAT session in the default reports directory of the current module.

Syntax:

```
public int GetFirstReport(    int Handle,
                              out int Report_Nr,
                              out string Report_Name,
                              out string Report_Filename );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Report_Nr`	returns the consecutive number of the report
`Report_Name`	returns the name of the report (for showing purposes)
`Report_Filename`	returns the filename of the report (for loading purposes)
`returns`	0 on success

3.8.2 GetNextReport

See above (`GetFirstReport`).

Syntax:

```
public int GetNextQueryName(  int Handle,
                              out int Report_Nr,
                              out string Report_Name,
                              out string Report_Filename );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Report_Nr`	returns the consecutive number of the report
`Report_Name`	returns the name of the report (for showing purposes)
`Report_Filename`	returns the filename of the report (for loading purposes)
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.8.3 GetReportName

This method returns the name and filename of a report with the given consecutive number.

Syntax:

```
public int GetReportName(     int Handle,
                              int Report_Nr,
                              out string Report_Name,
                              out string Report_Filename );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Report_Nr`	the consecutive number of the report
`Report_Name`	returns the name of the report (for showing purposes)
`Report_Filename`	returns the filename of the report (for loading purposes)
`returns`	0 on success

3.8.4 ReportFileName2Name

This method returns the name of a report with a given filename.

Syntax:

```
public int ReportFileName2Name (    int Handle,
                                    string Report_FileName,
                                    out string Report_Name );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Report_Nr`	the consecutive number of the report
`Report_Name`	returns the name of the report (for showing purposes)
`Report_Filename`	returns the filename of the report (for loading purposes)
`returns`	0 on success

3.8.5 GetReportExt

This method retrieves the report specified by the given data in PDF format.

Syntax:

```
public int GetReportExt(      int Handle,
                              string PrintReport,
                              int Part_Nr_x,
                              int Char_Nr_x,
                              int Value_Nr_x,
                              int NrOfValues_x,
                              int Part_Nr_y,
                              int Char_Nr_y,
                              int Value_Nr_y,
                              int NrOfValues_y,
                              int page_nr,
```


q-das@q-das.de • Tel. 06201/3941-0

```
                                    int Width,
                                    int Height,
                                    int Config_id,
                                    int OutputFormat,
                                    out string ReportStr);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`PrintReport`	the report filename
`PartNr_x`	the index number of the part, if the report has to be created for one characteristic (or a pair of characteristics) only, otherwise 0
`CharNr_x`	the index number of the characteristic, if the report has to be created for one characteristic (or a pair of characteristics) only, otherwise 0
`ValueNr_x`	0 (not used)
`NrOfValues_x`	0 (not used)
`PartNr_y`	the index number of the second part, if the report has to be created for a pair of characteristics only, otherwise 0
`CharNr_y`	the index number of the second characteristic, if the report has to be created for a pair of characteristics only, otherwise 0
`ValueNr_y`	0 (not used)
`NrOfValues_y`	0 (not used)
`page_nr`	0 to create all pages of a report, or a page number (> 0) to create a single page of a report
`Width`	the desired width of the graphic
`Height`	the desired height of the graphic
`Config_id`	0 or the key number of another graphic where the desired graphic can be embedded; in this case the graphic is configured like it would be in qs-STAT when embedded in the other graphic.
`OutputFormat`	0 – Bitmap; 1 – JPEG; 2 – Enhanced Metafile; 3 – Metafile
`ReportStr`	returns the string representation of an XML element containing the graphic (see description in `GetGraphic`)
`Returns`	0 on success

3.8.6 GetReportPages

This method returns the number of pages a given report would produce in the current session.

Syntax:

```
public int GetReportPages(    int Handle,
                              string PrintReport,
                              out int NrOfPages );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`PrintReport`	the file name of the report (without path)
`NrOfPages`	returns the number of pages the report would produce
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

g-das@q-das.de • Tel. 06201/3941-0

3.9 Commands to handle Catalogues

3.9.1 GetFirstCatalog

Together with `GetNextCatalog` this method can be used to create an enumeration of available catalogues. For this purpose after a successful call of `GetFirstCatalog`, `GetNextCatalog` is called as long as it returns 0.

Syntax:

```
public int GetFirstCatalog(   int Handle,
                              out int CatalogKey,
                              out string ColumnKeyList,
                              out int SubCatalogs,
                              out string CatalogName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`CatalogKey`	returns the key number of the first catalogue
`ColumnKeyList`	returns a comma-separated list of key numbers which represent the available columns
`SubCatalogs`	returns the number of subordinate catalogues if available
`CatalogName`	returns the name of the catalogue
`returns`	0 on success

3.9.2 GetNextCatalog

This method can be used together with `GetFirstCatalog`, see above.

Syntax:

```
public int GetNextCatalog(    int Handle,
                              out int CatalogKey,
                              out string ColumnKeyList,
                              out int SubCatalogs,
                              out string CatalogName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`CatalogKey`	returns the key number of the catalogue
`ColumnKeyList`	returns a comma-separated list of key numbers which represent the available columns
`SubCatalogs`	returns the number of subordinate catalogues if available
`CatalogName`	returns the name of the catalogue
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

3.9.3 GetFirstSubCatalog

Together with `GetNextSubCatalog` this method can be used to create an enumeration of available subordinate catalogues. For this purpose after a successful call of `GetFirstSubCatalog`, `GetNextSubCatalog` is called as long as it returns 0.

Syntax:

```
public int GetFirstSubCatalog(      int Handle,
                                    int CatalogKey,
                                    out int SubCatalog,
                                    out string SubCatalogName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`CatalogKey`	the key number of the main catalogue
`SubCatalog`	returns the key number of the first subordinate catalogue
`SubCatalogName`	returns the name of the subordinate catalogue
`returns`	0 on success

3.9.4 GetNextSubCatalog

This method can be used together with `GetFirstSubCatalog`, see above.

Syntax:

```
public int GetNextSubCatalog( int Handle,
                              int CatalogKey,
                              out int SubCatalog,
                              out string SubCatalogName );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`CatalogKey`	the key number of the main catalogue
`SubCatalog`	returns the key number of the subordinate catalogue
`SubCatalogName`	returns the name of the subordinate catalogue
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.9.5 GetFirstCatalogEntry

Together with `GetNextCatalogEntry` this method can be used to create an enumeration of available catalogue entries. For this purpose after a successful call of `GetFirstCatalogEntry`, `GetNextCatalogEntry` is called as long as it returns 0.

Syntax:

```
public int GetFirstCatalogEntry(    int Handle,
                                    int CatalogKey,
                                    int SubCatalog,
                                    int ColKey,
                                    out int EntryKey,
                                    out string OutStr );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`CatalogKey`	the key number of the main catalogue
`SubCatalog`	the key number of the subordinate catalogue
`ColKey`	the key number of the catalogue column
`EntryKey`	returns the key number of the entry
`OutStr`	returns the contents of the entry, specified by the column key number
`returns`	0 on success

3.9.6 GetNextCatalogEntry

This method can be used together with `GetFirstCatalogEntry`, see above.

Syntax:

```
public int GetNextCatalogEntry(     int Handle,
                                    int ColKey,
                                    out int EntryKey,
                                    out string OutStr );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`ColKey`	the key number of the catalogue column
`EntryKey`	returns the key number of the entry
`OutStr`	returns the contents of the entry, specified by the column key number
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.9.7 GetAnotherColumn

After a call of `GetFirstCatalogEntry` or `GetNextCatalogEntry` (see above), this method can be used to receive the contents of additional columns.

Syntax:

```
public int GetAnotherColumn(  int Handle,
                              int ColKey,
                              out int EntryKey,
                              out string OutStr );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`ColKey`	the key number of the catalogue column
`EntryKey`	returns the key number of the entry
`OutStr`	returns the contents of the entry, specified by the column key number
`returns`	0 on success

3.9.8 GetFirstCatalogEntryComplete

Together with `GetNextCatalogEntryComplete` this method can be used to create an enumeration of available catalogue entries, similar to `GetFirstCatalogEntry`, but returning all available fields of an entry. For this purpose after a successful call of `GetFirstCatalogEntryComplete`, `GetNextCatalogEntryComplete` is called as long as it returns 0.

Syntax:

```
public int GetFirstCatalogEntryComplete(  int Handle,
                                          int CatalogKey,
                                          int SubCatalog,
                                          out int EntryKey,
                                          out string NumberStr,
                                          out string NameStr,
                                          out string String3,
                                          out string String4,
                                          out string String5,
                                          out string String6,
                                          out string String7,
                                          out string String8 );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`CatalogKey`	the key number of the main catalogue
`SubCatalog`	the key number of the subordinate catalogue
`EntryKey`	returns the key number of the entry
`NumberStr`	returns the number field of the entry (column key K4XX2)
`NameStr`	returns the description field of the entry (column key K4XX3)
`String3..String8`	return further fields of the entry
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.9.9 GetNextCatalogEntryComplete

This method can be used together with `GetFirstCatalogEntryComplete`, see above.

Syntax:

```
public int GetNextCatalogEntryComplete(   int Handle,
                                          out int EntryKey,
                                          out string NumberStr,
                                          out string NameStr,
                                          out string String3,
                                          out string String4,
                                          out string String5,
                                          out string String6,
                                          out string String7,
                                          out string String8 );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`EntryKey`	returns the key number of the entry
`NumberStr`	returns the number field of the entry (column key K4XX2)
`NameStr`	returns the description field of the entry (column key K4XX3)
`String3..String8`	return further fields of the entry
`returns`	0 on success

q-das@q-das.de • Tel. 06201/3941-0

3.10 Other commands

3.10.1 DoAction

This method can be used to call an internal function of qs-STAT.

Syntax:

```
public int DoAction(    int Handle,
                        int Fkt_group,
                        int Fkt_nummer,
                        int Fkt_sub_nummer,
                        int Fkt_zus_nummer,
                        out string Key_info_out );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Fkt_group`	the group key, function key, subkey and additional key of the function to be called
`Fkt_nummer`	
`Fkt_sub_nummer`	
`int Fkt_zus_nummer`	
`Key_info_out`	might return information, depending on the function called
`returns`	0 on success

3.10.2 DoAction_Ext

Like `DoAction` this method can be used to call an internal function of qs-STAT. In addition to the latter it can handle additional information provided by an XML string in `Parameters`.

Syntax:

```
public int DoAction(    int Handle,
                        int Fkt_group,
                        int Fkt_nummer,
                        int Fkt_sub_nummer,
                        int Fkt_zus_nummer,
                        string Parameters,
                        out string Key_info_out );
```

Parameters:

`Handle`	the handle of the qs-STAT session
`Fkt_group`	the group key, function key, subkey and additional key of the function to be called
`Fkt_nummer`	
`Fkt_sub_nummer`	
`int Fkt_zus_nummer`	
`Parameters`	an XML structure containing additional information
`Key_info_out`	might return information, depending on the function called
`returns`	0 on success

g-das@q-das.de • Tel. 06201/3941-0

XML structure of `Parameters` *(example):*

```
<DoAction>
      <Parameters>
            <ControlInfo senderKey="5200" graphicKey="3100"/>
            <ParamZu1 type ="int" value="123"/>
            <ParamZu2 type ="double" value="1.23"/>
            <ParamZu3 type ="bool" value="1"/>
            <ParamZu4 type ="string" value="Some text"/>
            <ParamZu5 type ="stringList">
                  <String value="First line"/>
                  <String value="Second line"/>
                  <String value="Third line"/>
            </ParamZu5>
            <ParamZu5 type ="qdasCharNode" partNo="1" charNo=”4”/>
            <KeyInfoOut/>
      </Parameters>
</DoAction>
```

3.10.3 GetGridInfo

This method can be used to receive information needed to fill an input grid, only used to display a VDA 5 input grid. The parameter `GridInfoXml` contains information about the grid such as the number of rows and columns, sizes of rows and columns, and the contents of cells.

Syntax:

```
public int GetGridInfo( int Handle,
                        int GridKey,
                        int GridSubKey,
                        string ParamXml,
                        out string GridInfoXml);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GridKey`	the key of the grid (until now only 12301)
`GridSubKey`	the subkey of the grid (7764 [MS], 7765 [MP] or 7769 [MPE])
`ParamXml`	an XML structure containing additional definitions (not yet used)
`GridInfoXml`	returns grid information as an XML structure
`returns`	0 on success

XML structure of `GridInfoXml` *(example):*

```
<Grid colCount="5" rowCount="15" fixedColCount="5" fixedRowCount="15"
      defaultColWidth="120" defaultRowHeight="22" focusCol="1" focusRow="1"
      cellsInColumns="1" >
      <Columns>
            <Col colId="0" readOnly="1" type="String" Width="150">
                  <Cell rowId="0" type="String" value="Some text"/>
                  <Cell rowId="1" type="String" value="Some new text"/>
                  .
                  .
```


g-das@q-das.de • Tel. 06201/3941-0

```
                .
            </Col>
            <Col colId="1" readOnly="0" type="Float" Width="150">
                  <Cell rowId="0" readOnly="1" type="String" value="text"/>
                  <Cell rowId="1" value="12.34"/>
                  .
                  .
                  .
            </Col>
            .
            .
            .
      </Columns>
      <Rows>
            <Row rowId="0" readOnly="1">
      </Rows>
</Grid>
```

3.10.4 SetGridInfo

This method can be used to send changed contents of an input grid back to the web service (up to now only supported: VDA 5 input grid). The parameter `GridInfoXml` contains the same structure as received in `GetGridInfo`, but with changed cell contents, depending on the changes the user has made.

Syntax:

```
public int SetGridInfo (      int Handle,
                              int GridKey,
                              int GridSubKey,
                              string ParamXml,
                              string GridInfoXml);
```

Parameters:

`Handle`	the handle of the qs-STAT session
`GridKey`	the key of the grid (until now only 12301)
`GridSubKey`	the subkey of the grid (7764 [MS], 7765 [MP] or 7769 [MPE])
`ParamXml`	an XML structure containing additional definitions (not yet used)
`GridInfoXml`	grid information/contents as an XML structure
`returns`	0 on success

3.10.5 SessionCount

This method returns the number of sessions which are connected to the web service.

Syntax:

```
public int SessionCount ();
```


q-das@q-das.de • Tel. 06201/3941-0

Parameters:

`returns`	the number of connected sessions

g-das@q-das.de • Tel. 06201/3941-0

4 Trouble Shooting

4.1 Problems on Accessing the Web Service

If the web service cannot be addressed by “localhost” (see “Getting Started”) or if it runs on a remote system, and your web application was created by Q-DAS Web Designer, the files “IQdas_Web_Serviceservice.wsdl” and “IQdas_Web_Serviceservice.discomap” must be changed manually: In both files the entry “http://localhost” has to be replaced by the correct address.

4.2 Web Application does not Keep Session Variables

It happened that the web application lost its session variables on each PageLoad (`Session["Info"]` always was `null`) so that it was not possible e.g. to show a graphic of a part previously displayed in a list. The problem only occurred with Internet Explorer, and it did not occur if the web server was addressed by its IP address. It also did not occur any longer after the server had been renamed (an underscore was removed). See http://classicasp.aspfaq.com/general/why-won-t-my-session-variables-stick.html, item #4.

4.3 Problems on Connection to Oracle Databases

If the web service and the Oracle database are installed on the same server, also an Oracle client must be installed on this server, and the connection (UDL file) must be created using a local net service name (the name of the database instance does not work in this case). The net service name must use an IP address instead of a host name.

g-das@q-das.de • Tel. 06201/3941-0