HALLER + ERNE GMBH

BS300IF Manual

Version 2.100 – 2009-02-02

User manual and reference for BS300IF DLL – Interface for easy access of the System 300 Low-Level communication interface.

VERSIONS

Document versions:

V1.0	2008-02-19	HE	First document release
V1.1	2008-03-25	HE	Added curve functions
V1.2	2008-04-21	HE	Added description for function return values Added function descriptions: BSIF_Crv_SaveCrv and BSIF_Crv_GetCrvData, Modified functions: BSIF_Tools_CopyObjData Functions marked obsolete: BSIF_Tools_SaveCurve, BSIF_Tools_GetCurveData, BSIF_JobResultData Added English translation
V1.3	2008-07-07	HE	Added function description: BSIF_Prg… tightening program functions BSIF_SetLicense Added licensing description Fixed some English translation errors
V1.4	2008-02-02	HE	Added function description: BSIF_App… tightening program functions BSIF_Val… actual value functions BSIF_Sys… system information functions Added description for new firmware V2.100, firmware autodetect mechanism Added description for new filtered curve request definitions/functionality Added description for new program/application list functions Added pseudocode samples **Breaking changes:** BSIFPrgData type changed to include dwLevel for program info level

This documentation is valid starting with BS300DLL V2.100.1.2.

TABLE OF CONTENTS

1 BS300IF OVERVIEW

1.1 OVERVIEW

This DLL provides an easy to use interface for communicating with the Bosch Rexroth tightening system. The DLL supports all tightening systems of the series 300/310/350 with firmware versions V1.300-V2.100 (depending on the version some features (curve buffer) might not be available). The DLL uses standard Windows calling conventions and provides header files and import libraries for Microsoft Visual C++ (coff) and Borland C/C++ (omf) (Microsoft.Net IJW Wrapper, Visual Basic, Delphi and Linux-Versions on request).

Additionally there are some sample projects for Microsoft Visual C/C++, which show how to use the DLL for different application areas.

1.2 APPLICATION AREAS

The DLL provides support for the following application areas (and probably much more):

- Tightening program transmission and access:
  - Backup and restore tightening programs and applications, e. g. implement a central tightening program database for a whole line
  - Circumvent the limit of 48 tightening programs per channel, e. g. build highly flexible assembly stations by downloading tightening programs to the tightening system based on the type of product to be assembled, …
  - Manipulate tightening programs, e. g. change parameters like speed, angle, torque, … e. g. use the DLL in National Instruments LabView to dynamically update tightening program parameters according to measured values.
  - Decode and analyze tightening programs, e. g. to store key parameters in a database or printout the tightening programs
  - Verify tightening programs and their versions, check e. g. for the last change date or checksum
- Tightening curve transmission and access:
  - Read the tightening curves from the tightening system and process them, e. g. for storing in a quality database, for reading key parameters from the curve or for converting them into a 3rd party data format.
  - Read and write tightening curves in BS300/BS350 compatible format (*.crv) to use the standard tools to visualize and analyze tightening curves.
- System information:
  - Scan the tightening system for a list of channels, applications and programs, e. g. automatically scan/detect unauthorized tightening program modifications.
  - Implement backup/restore type of application.
  - Use the system information to configure 3rd party applications, i. e. wizards for mapping tightening result data (application/program) to tightening tool databases.
  - Get information on the firmware version of the tightening controller to implement application functionality which works with all firmware versions.
  - Read serial numbers of tools and controllers for use with inventory/asset management or maintenance databases.
- Actual value transmission:
  - Implement diagnostics and measurement applications requiring realtime tightening result data without changing the tightening systems configuration.
  - Implement tool verification applications like machine capability (MFU) or process capability (PFU) testers, running without impact on the target tightening system or its configuration.

To implement these applications, the DLL provides a set of simple to use functions and data structures:

- Tightening program transmission and access:
  - Read and write tightening program files in BS300/BS350 compatible format (*.prx, *.prg) from/to disk.
  - Upload tightening programs from the PC into the tightening system.
  - Download tightening programs from the tightening system to the PC.
  - Decode and analyze tightening programs, read and change tightening parameters.
- Tightening curve transmission and access:
  - Read and write tightening curves in BS300/BS350 compatible format (*.crv) from/to disk.
  - Download tightening curves from the tightening system (actual curves or curves from internal curve buffer, optionally filtered)
  - Decode and analyze tightening curves and convert them in a version-independent, stable and easy to use data format.
- System information:
  - Read the list of tightening system modules of which the tightening system consists (get their names and physical positions)
  - Get a list of existing tightening applications for a KE (application sequence number and name)
  - Get a list of existing tightening programs for a SE/CS (program number and name)
  - Get tightening channel and tool information (serial numbers)
- Actual value transmission and access:
  - Read the list of tightening system modules of which the tightening system consists (get their names and physical positions)

All these functions aim to work firmware independent. The BS300IF-DLL automatically detects the tightening system type and firmware version and automatically generates the correct communication packets. It also converts any data received from the tightening system into version-independent data structures, so the application programmer does normally not need to handle different firmware versions for the most common operations.

If newer tightening system firmware versions provide more data, a new data structure with extended information gets added to the BS300IF-DLL. This means the application can work with a single data structure across all firmware version for a given data type – if it chooses to only use the basic, common set of information. If extended information (which might not be available for all firmware versions) is required, it can ask the API to provide a higher “information level” specific for the newer firmware version.

1.3 TERMS

The following terms are important for understanding the functions inside the BS300IF-DLL:

- Communication channel: This term describes a communication link between the BS300IF-DLL and the tightening system. A communication channel describes either a single TCP/IP-connection or a serial Interface connection. By using a single communication channel the BS300IF-DLL can communicate with multiple tightening channels (in this case a KE inside the tightening system is required).
- Tightening channel: This term describes a physical unit in the tightening system. A tightening channel usually consists of a tool, a tightening control unit (SE) and the cable between the tool and the SE/SE3xx). A tightening channel can be uniquely identified inside a tightening cell by its Channel number (or channel address) in the form:

  Ch<Rack>.<Slot> (e. g. Ch3.1)

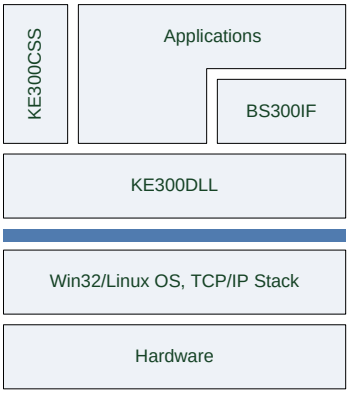
<Rack> is physically given by the rack-number (rotary switch on the backplane of the system box or rack enclosure) to which the tightening channel is connected; <Slot> is the physical slot number where the tightening system controller (SE/SE3xx) is plugged into the box/rack.

- Tightening cell: This term descries all tightening channels which are connected to a single KE. If a tightening channel is not connected to a KE then this tightening channel is a tightening cell on its own.
- Tightening system: The whole system might consist of multiple tightening cells.

All additional terms are used as defined in the tightening system documentation.

1.4 STRUCTURE OF THE BS300IF.DLL

The following picture shows the structure of the BS300IF.dll:



The low-level communication with the tightening system is handled by the KE300DLL. This layer implements special handling for different Tightening system firmware and protocol versions and also provides hardware-independent communication interfaces over TCP/IP and serial ports. The DLL supports handling of timeouts and automatic reconnects and provides a platform independent interface for Win32 and Linux.

The next higher level is implemented in the BS300IF-DLL. This DLL provides “High-Level“ communication services with an easy to use API, e. g. “download tightening program parameters“. In addition to this, the DLL contains a set of utility functions, e. g. to read and write tightening system objects in the standard file format of the system. This can be used to import and export files from/to BS300 (the official tightening system configuration software).

The KE300DLL uses a generic model of a communication interface. In general this has one or more communication channels. Each communication channel provides one or more virtual connections between the communication endpoints. Each virtual connection can handle one or more jobs. A job is an object, which has state information and is used to exchange data between the communication partners. The KE300DLL can in general handle multiple active jobs per virtual connection, but in case of the System 300/310 tightening controllers this is not necessary (see below).

To make this communication model easier to use, the BS300IF handles some of the complexity internal and also makes some simplifications. The BS300IF handles all jobs internally and only uses exactly one job for each tightening channel. This means, that at one time only one communication transaction for a tightening channel can be active. However, this is no severe restriction, as the tightening system itself only allows one command to be transmitted for most of the available commands (request/response command system). As in addition to this all jobs are handled internally, the user of the BS300IF-DLL does not need to create/destroy job objects.

1.5 LICENSING

To use all the features of the DLL a license is required. If no valid license is available, then the DLL works in demo mode. There are different licensing models available: node locked and user locked:

- Node locked license:

  This license type locks the DLL to a computer system. For each computer a unique license is required. Licensing works by registering the DLL after installing it onto the target computer. The procedure is as follows:

1. Install the DLL on the target system
2. Run the registration wizard and enter the user information. The wizard will generate a unique registration key for the system.
3. Send the generated registration key to us by email.
4. We validate the registration key and send you a license key back.
5. Enter the license key into the registration wizard to unlock the DLL (or put the license file into the installation directory of the DLL).

- User locked license:

  This license type locks the DLL to a customer and requires using the DLL API to provide a customer specific serial key at runtime. The procedure is as follows:

  1. Order a user locked license for the DLL
  2. We generate a unique customer specific serial key and a license file. Both, license file and serial key are sent to you by email.
  3. Modify your application which uses the DLL to call the API function BSIF_SetLicense() passing your serial key as a parameter before calling BSIF_Init() (see below for details).
  4. Install the DLL on the target system and copy the license file into the installation directory of the DLL.

2 FUNCTION OVERVIEW

2.1 LIST OF FUNCTIONS

The following list gives an overview of the functions available in the BS300IF-DLL, grouped by application area. A more detailed description is available in the function reference in the next chapter.

Functions for initialization and shutdown

BSIF_Init	Initializes the DLL and provides a handle for further access of the API.
BSIF_SetLicense	Passes licensing information to the DLL.
BSIF_Shutdown	Stops all communication and releases all allocated resources.

Functions for configuration

BSIF_AddKE	Adds a new communication channel (TCP/IP or serial) to the configuration
BSIF_AddSE	Add a new tightening channel to the configuration
BSIF_DelKE	Deletes a communication channel
BSIF_DelSE	Removes a tightening channel from the configuration

Generic functions for managing and accessing jobs

BSIF_JobState	Read the current state of a job
BSIF_JobDone	Checks, if a job is completed („done“).
new BSIF_JobSetTimeout	Set a jobs timeout value
BSIF_Tools_CopyObjData	Copies job result data (payload) received from the tightening system into a user-provided buffer.

Tightening system information

new BSIF_JobAdd_BP_LOGONREQ	Read the list of connected tightening system modules
new BSIF_JobAdd_BP_SYS_APPNAMTAB	Read the list of defined tightening applications
new BSIF_JobAdd_BP_SYS_PRGNAMTAB	Read the list of defined tightening programs
new BSIF_JobAdd_BP_Diag	Request diagnostic information from a tightening system component, e. g. serial numbers…
new BSIF_Sys_GetLogonData	Decode the logon object and provide an easy to handle structure for accessing the list of tightening system modules
new BSIF_Sys_GetAppNameTbl	Decode the list of tightening applications of a KE and provide an easy to handle structure for accessing it
new BSIF_Sys_GetPrgNameTbl	Decode the list of tightening programs of a tightening channel and provide an easy to handle structure for accessing it
new BSIF_Sys_GetDiagData	Decode diagnostic information and provide an easy to use structure for accessing it.

Functions for tightening programs and parameters

BSIF_JobAdd_BP_READ_SRBPROG	Add new job for downloading (tool → PC) a tightening program and start the job immediately.
BSIF_JobAdd_BP_WRITE_SRBPROG	Add a new job for uploading (PC → tool) a tightening program and start the job immediately
new BSIF_JobAdd_BP_DEL_ALL_PRG	Add a new job to delete all tightening programs of a channel and start the job immediately
new BSIF_JobAdd_BP_READ_SRBAPP	Add new job for downloading (tools KE → PC) a tightening application definition and start the job immediately.
new BSIF_JobAdd_BP_WRITE_SRBAPP	(not implemented at the moment)

BSIF_JobAdd_BP_DEL_ALL_APP	(not implemented at the moment)
BSIF_Prg_ReadPrg	Read a tightening program object from a prg-file.
BSIF_Prg_ReadPrx	Read a tightening program object from a prx-file.
BSIF_Prg_SavePrg	Save a tightening program object into a prg-file.
BSIF_Prg_GetPrgData	Decodes a tightening program object and provides an easy to handle structure for accessing the tightening program information.
BSIF_Prg_SetParam	Modifies a parameter value of a tightening program object.

Functions for tightening curves

BSIF_JobAdd_BP_CURVE	Add a new job to request tightening curve. Uses default parameter values for curve request type and operation mode.
BSIF_JobAdd_BP_CURVE_Ext	Add a new job to request tightening curve. Allows to provide user defined parameters for the curve request mode and type.
BSIF_Crv_SaveCrv	Save curve as BS300 .crv-file.
BSIF_Crv_GetCrvData	Decodes a curve object and provides an easy to handle structure for accessing the curve data.

Functions for actual value access

BSIF_JobAdd_BP_SEND_TBLRES_SE	Request the actual value table of a channel
BSIF_JobAdd_BP_SEND_TBLRES_KE	Request the actual value table of a KE
BSIF_Val_SetCallback	Setup callback function for immediate callback on new tightening result
BSIF_Val_GetSEData	Decode a actual value object to easily use its data

Additional functions:

BSIF_GetKEState	Returns information on the current communication state of the communication channel (TCP/IP-connection or serial connection).

Obsolete functions (still supported, but should not be used any more):

BSIF_Tools_SaveCurve	Save curve into .crv file. Obsolete, use BSIF_Tools_CopyObjData and BSIF_Crv_SaveCrv instead
BSIF_Tools_GetCurveData	Decodes curve data. Obsoltete, use BSIF_Tools_CopyObjData and BSIF_Crv_GetCrvData instead.
BSIF_JobResultData	Provides access to the jobs internal data buffer (payload). Obsolete, use BSIF_Tools_CopyObjData instead.

2.2 GENERAL OPERATION

2.2.1 INITIALIZATION

Before using any of the communication functions from the BS300IF-DLL, the system needs to be configured first. Therefore the communication parameters need to be defined and the tightening channels must be added to the configuration.

The general sequence is as follows:

1. (only for user locked licenses) Initialize Licensing by calling the function BSIF_SetLicense().
2. Initialize the BS300IF-DLL by calling the function BSIF_Init().
3. Call the function BSIF_AddKE() for each communication channel (normally KE). Important: define a unique iKEID for each call!
4. Call the function BSIF_AddSE() for each tightening channel to add it to the configuration.

After the communication parameters are set and the tightening channels are added, the BS300IF-DLL automatically starts the communication.

2.2.2 TIGHTENING CHANNEL ADDRESSING

A tightening channel within a tightening cell is uniquely addressed by a channel number (address) of the following form:

Ch<Rack>.<Slot> (z. B. Ch3.1)

<Rack> is physically given by the rack-number (rotary switch on the backplane of the system box or rack enclosure) to which the tightening channel is connected; <Slot> is the physical slot number where the tightening system controller (SE/SE3xx) is plugged into the box/rack.

Inside the DLL this address is converted to a number, which is calculated from <Rack> and <Slot>. To convert a given <Rack> and <Slot> number to the unique channel number, use the makro RS() from the header file bs300if_common:

<unique channel number> = RS(<Rack>, <Slot);

This unique channel number is used in all BS300IF-functions requiring a channel number parameter (parameter iSEChn).

2.2.3 EXECUTING A JOB

The pattern for using the BS300IF-DLL is as follows:

1. Initialize job and hand it over to the DLL. Therefore the BS300IF-DLL provides the functions BSIF_JobAdd_XXX. E. g. to request (download) a tightening program parameter set from the tightening controller to the PC use the function BSIF_AddJob_BP_READ_SRBPRG.
2. Wait until job processing is completed. Note that the DLL guarantees, that each and every job submitted to the DLL completes (by completing OK/correctly, by error or by abort/timeout). To check for „job complete“, use the function BSIF_JobDone and/or BSIF_JobState (returns additional job status information).
3. If a job has completed, then the job might contain result data (depending on the type of job or requested information). To access this result data block, use BSIF_Tools_CopyObjData (BSIF_JobResultData) and (if necessary) access/convert it using special functions (e. g. for tightening programs BSIF_Prg_xxx - functions).
4. Continue with step 1.

In the sample application for modifying tightening program parameter values (ModProg.cpp) this pattern is used many times to run through the following steps:

1. Download a tightening program from the tightening controller to the PC (using the pattern shown above)
2. Print some information about the tightening program
3. Change a parameter value (change RPM in step 2A)
4. Upload the modified tightening program from the PC to the tightening controller (using the pattern shown above)
5. Download the same tightening program again (like in Step 1, using the pattern shown above)
6. Compare the modified tightening program with the program downloaded from the system to check if the parameter modification was successful.

2.3 CURVE TRANSMISSION

The tightening system provides many ways to transmit tightening curves. Besides different curve transmission parameters (type of curve (e. g. Angle/Torque or Time/Angle/Torque, …), filters, etc.) there are also different curve transmission modes. The following two modes are important:

- Realtime curve transmission: In this case the PC sends a “subscribe”-command to the tightening controller. The tool will respond as soon as there is a new curve available (after a new tool run). After receiving a new curve, the subscription needs to be reissued. Using this mode, curves will be transmitted as fast as possible (“realtime”). This mode is also called “publish/subscribe” mode.
- Curve buffer access: Using this mode, the PC can request a curve with a given (unique per channel) sequence number from the channels internal curve buffer. This mode is only available for newer generation firmware and must be enabled using the BS300 tightening system configuration application. This mode also allows to access historical curves (in case the network is down).

Depending on the curve transmission mode, the communication sequence is a bit different. The following chapters describe the required sequence and behavior in more details.

2.3.1 PUBLISH/SUBSCRIBE

The publish/subscribe system for requesting tightening result curves is the most easy system to get curves out of the tightening controller. The sequence follows the standard-pattern (see 2.2.3) closely, but there is an additional sequence-parameter:

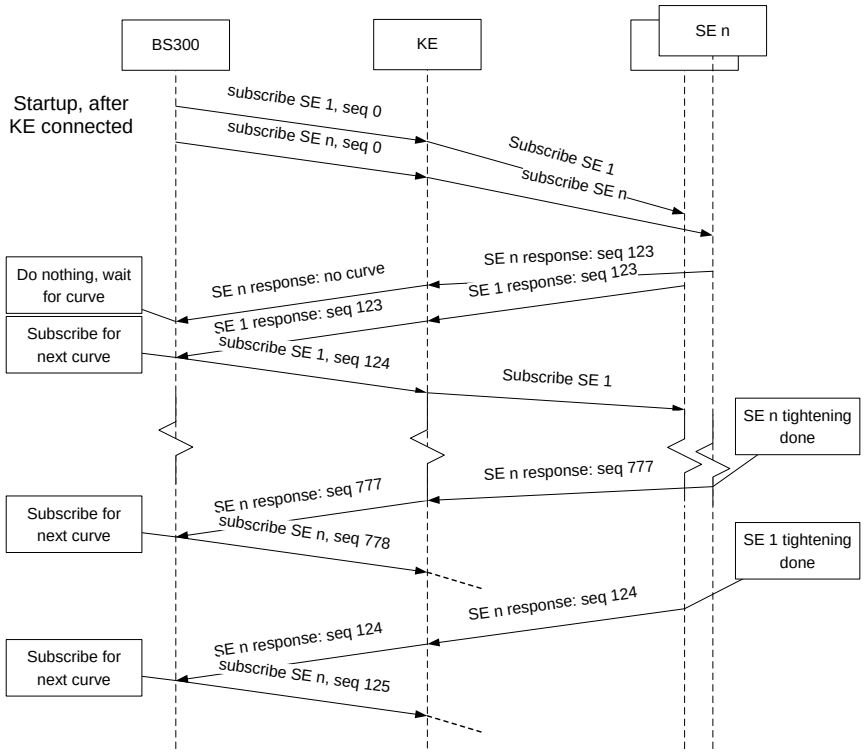
1. Create a new variable for storing the tightening channels current sequence number and initialize it with zero. Zero means to request the “current” curve, independent from the actual sequence number of the channel. This allows the PC to start the publish/subscribe system without knowing the sequence number a priori.
2. Init job and hand it over to the DLL. In the simplest case, use the function BSIF_JobAdd_BP_KURVE and use the sequence number variable defined in step 1) as parameter. If additional parameters should be used (e. g. filter criteria) then use the function BSIF_JobAdd_BP_KURVE_Ext instead.
3. Wait until the job processing is completed. Note that the DLL guarantees, that each and every job submitted to the DLL completes (either by completing OK/correctly, by error or by abort/timeout). To wait check for „job complete“, use the function BSIF_JobDone and/or BSIF_JobState (returns additional job status information).

   For curve transmission the following cases need to be checked:

   - The tightening controller has no curve (not yet). This happens after powerup of the tightening controller. In this case, the sequence number must stay zero and the request needs to be reissued (continue with step 2))
   - There is a new curve available. In this case, the curve data can be decoded (BSIF_Tools_GetCurveData) and processed. To prepare the next curve request subscription, the channels sequence number variable (step 1) should be updated with the sequence number returned from the tightening system (BSIFCurveData->Sequence). Then the request should be reissued with the updated sequence number (continue with step 2).
   - There is not yet a new curve available. This state is indicated by the job having a state of HEJS_Done_WTO. In this case the current sequence number variable should not be changed and the job should be resubmitted (continue with step 2). This procedure makes sure that in special cases (e. g. reset of a SE without reset of the KE) the PC resubmits a subscription command to the SE/KE as the SE/KE might loose curve subscriptions in these cases.

4. If there is an error, then continue with step 1 (reset sequence counter to zero).

As a summary, the following sequence diagram shows the general sequence of events with sample values for different channels:



When using the publish/subscribe system, the following points need special attention:

- Realtime curve transmission might (especially when using KE/FO application start) cause delays in the tightening cycle (CyCmp-signal of FO x). The KE will set the FOx.CyCmp signal after all tightening channels with active curve subscription have transmitted their curve data to the KE using the internal ArcNet bus.
- If the network is down or a channel is rebooted, curve data might get lost (online-operation mode)

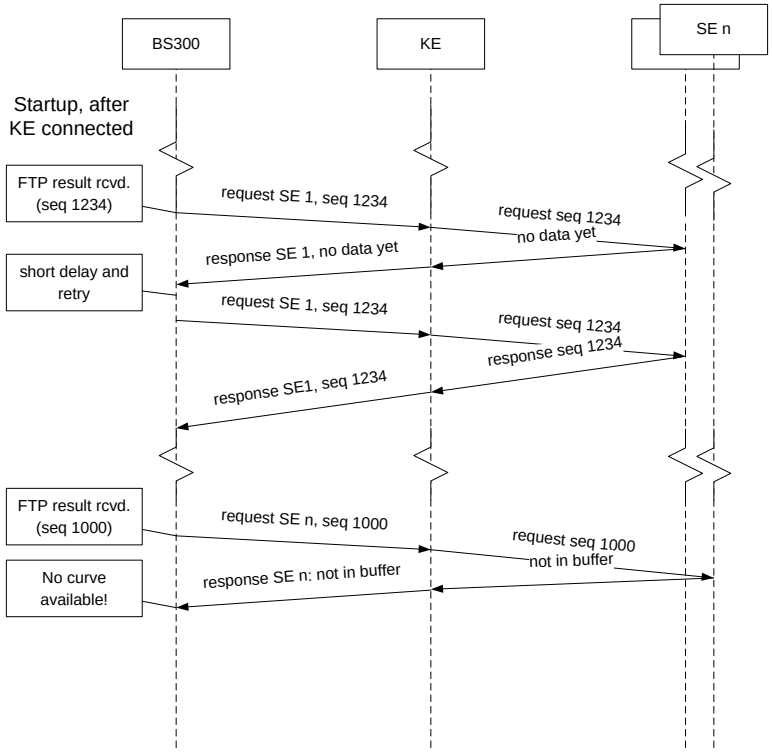
2.3.2 CURVE BUFFER

Requesting curves from the curve buffer also follows the standard pattern for submitting a job and waiting for job completion (see 2.2.3), but the sequence number of the curve to be requested must be known a priori. This curve request mode is convenient when e. g. using it in combination with the FTP data output of the tightening system (which delivers the current channels sequence number). The actual sequence is as follows:

1. Initialize job with sequence number of curve to be requested and hand it over to the DLL. To use the curve buffer the function BSIF_JobAdd_BP_KURVE_Ext must be used and the parameter BSIFCurveParams.bMode must be set to 0x08 (buffer mode). The sequence number must be known before calling the function (e. g. from FTP-tightening result data block).
2. Wait until the job is completed (see above).
3. Check the jobs result state and curve object response status code. The following cases are possible:
   - The requested sequence number is not yet available. In this case, the request should be resubmitted after a short delay (with the same sequence number; continue with step 1).

- The requested sequence number is not available any more (too “old”). In this case the curve is not available and will never also. The sequence number is therefore invalid, so a new sequence number must be used for the next cycle.
- Curve data is available. Decode it and process it (BSIF_Crv_GetCrvData).
- There are errors (curve buffer not active).

As a summary, the following sequence diagram shows the general sequence of events with sample values for different channels (sequence number from FTP result):



When using the curve buffer system, the following points need special attention:

- Curve buffer is only available in new tightening systems with recent hardware (>= System 310) and firmware and must be enabled using the BS300/BS350 system configuration tool.
- You can only request curves, which are stored inside the channels curve buffer – this means the curve buffer configuration defines which curves are available at all (BS300/BS350 allows to set filters for program numbers, OK/NOK results and maximum number of points), the curve request issued using the DLL can only request additional filtering. If e.g. the curve buffer is configured to store only NOK curves, requesting an OK curve will never return a curve!
- Additional external sequence number information is required

2.4 RETURN VALUES

The functions of the BS300IF-DLL return the following values:

HE_E_WIN32ERROR	Operating system error (Win32- or WinSock error) occured. Use GetLastError() to get additional error information. Errors from the low-level communication (handled inside the KE300DLL) are always encoded as WIN32-errors. Common errors returned by GetLastError(): - ERROR_INVALID_PARAMETER (wrong parameter) - ERROR_INVALID_ENVIRONMENT(API not intialized)

	- ERROR_INVALID_HANDLE (channel number invalid)
HE_E_BUFTOOSMALL	Buffer supplied as an argument is too small
HE_E_INVALIDARG	Invalid argument passed to the function
HE_E_INVALIDOBJ	A pointer passed to the function points to an invalid object or the referenced memory contains invalid data.
HE_E_INVALIDVER	Invalid version or firmware version.
HE_E_SIZEMISMATCH	Mismatch in size. Normally this means an object passed to a function is corrupt or a size-parameter given as argument is wrong.
HE_E_CRCERROR	Error checking the CRC.
HE_E_UNCOMPRESS	Error uncompressing a data block.
HE_E_FILEREAD	Error while reading a file.
HE_E_INVALIDHDR	Invalid structure or header.
HE_E_INVALIDADR	Invalid address or index.
HE_E_NOTFOUNDKE	KEID could not be found in internal list of communication channels (need to call AddKE for each KE/KEID)
HE_E_IDNOTUNIQUE	The given ID (KEID and or channel number) is not unique. Do not call AddSE or AddKE with the same parameters twice.
HE_E_NOTFOUNDID	The given ID (KEID and or channel number) is not registered (need to call AddKE od AddSE before calling other functions).
HE_E_JOBNOTRDY	The function call is not allowed at the moment, as the job is not in the idle or done state. The most common cause is not waiting for the job to be completed before starting a new job (check JobDone() before trying to start a new job)
HE_E_INVALIDPRG	Invalid program number or program number is not available in the prx-file.
HE_E_NOTIMPL	Function is not implemented at the moment

3 REFERENCE

3.1 INITIALIZATION AND SHUTDOWN

3.1.1 BSIF_Init

This function initializes the BS300IF-DLL and returns a handle for further API access.

```
DWORD BSIF_Init (
    const char* pszIniFile
);
```

Parameters

pszIniFile
[in] Unused in the current API revision and must be NULL.

Return Values

A return value of zero indicates an error. In this case GetLastError() can be used to get a detailed error code. In any other case the return value is a handle for use with calling the other API functions.

Remarks

This function must be called after BSIF_SetLicense (if used) and prior to all other functions in this DLL.

3.1.2 BSIF_SetLicense

This function sets a property value in the licensing interface.

```
DWORD BSIF_SetLicense (
    const char* pszLicPropName,
    const char* pszLicPropValue
);
```

Parameters

pszLicPropName
[in] License property name. At the moment only “UserKey” is allowed.

pszLicPropValue
[in] License property value to set.

Return Values

A return value of zero indicates success (property value set successfully). A return value other than zero indicates an error (see chapter 2.4 for details).

Remarks

If a user-locked license is used, then this function must be called prior to all other functions in the DLL. In this case the function must be called with the property name “UserKey” and the property value string set to the unique user key received with the license file.

3.1.3 BSIF_Shutdown

This function stops all communication and frees all resources.

```
DWORD BSIF_Shutdown (void);
```

Parameters

This function has no parameters.

Return Values:

A return value of zero indicates success. Else an error code is returned (see 2.4).

3.2 CONFIGURATION

3.2.1 BSIF_AddKE

This function adds a new communication channel and starts the communication.

```
DWORD BSIF_AddKE (
    DWORD dwHandle,
    int iKEID,
    const char* pszAddr,
    int iFirmwareVersion
);
```

Parameters

dwHandle

[in] Handle returned by calling BSIF_Init.

iKEID

[in] Unique ID for identifying the communication channel. This parameter can be chosen by the user, the only requirement is that it must be unique per KE. When calling other API functions this value identifies the communication channel to use.

pszAddr

[in] Pointer to a ASCIIZ-string containing a remote communication endpoint address. At the moment two types of addresses are possible: TCP/IP and serial. To use a TCP/IP connection, this parameter contains the hostname or the dotted numeric IP-Adresse of the tightening system. For using a serial connection, the complete Windows-devicename in the form \\.\COMx must be used instead (e. g. in C/C++: "\\\\.\\COM1").

iFirmwareVersion

[in] Version of the communication protocol to use. The following table shows the possible values:

Wert	Beschreibung
3	Firmware V1.300 (not all functions are available)
4	Firmware V1.350
5	Firmware V1.450 (GM Common Controller, not all functions are available)
6	Firmware V1.500/V1.600

Return Values

A return value of zero indicates success (communication parameters successfully initialized). However, this does not indicate the communication is actually possible or running. To get information on the current communication state, use BSIF_GetKEState.
A return value other than zero indicates an error (see chapter 2.4 for details).

Remarks

A communication channel is either a TCP connection or a serial connection (not all commands are supported for serial communication) with the tightening system. In general the behavior is different for System 300 and System 350: For System 300/310 a TCP connection is only possible with a KE, with System 350 a TCP connection is also possible for a single tightening channel (CS351).
The DLL starts a new thread for each communication channel.

3.2.2 BSIF_DelKE

This function stops an existing communication channel and removes it from the internal configuration.

```
DWORD BSIF_DelKE (
    DWORD dwHandle,
    int iKEID
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

If the communication channel has active jobs when calling this function, then they are aborted (BSIF_Job_Abort) and deleted (BSIF_Job_Delete). The communication channel will be stopped and all allocated resources will be destroyed. This might take some times depending on the job and communication timeouts (max. 500ms).

3.2.3 BSIF_AddSE

This function adds a tightening channel to the internal configuration.

```
DWORD BSIF_AddSE (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    const char* pszName
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

pszName
[in] Pointer to an ASCIIZ-string containing the name of the tightening channels. This information is only used for debugging/logging.

Return Values

A return value of zero indicates success (parameters successfully initialized). However, this does not indicate the communication with the channel is actually possible or running. Normally communication with a tightening channel is only active after a job has been started by calling one of the functions BS300IF_JobAdd_xxx().
A return value other than zero indicates an error (see chapter 2.4 for details).

Remarks

Calling this function creates, allocates and initialized an internal job object for communicating with a tightening channel. In general the communication system of the KE300DLL supports multiple jobs for the same tightening channel, but the BS300IF-DLL always uses only one per tightening channel. The communication in regards to a single channel is therefore quasi-half-duplex (quasi, as there are exceptions, but they are handled internally), so at one time there is at most one job active per tightening channel. The By using this simplification, the BS300IF-DLL does not need to allocate memory dynamically during execution and all KE300DLL job management functions can be implemented internally.
Before calling this function a communication channel must be defined by calling BSIF_AddKE.

3.2.4 BSIF_DelSE

This function stops all active communication with a tightening channel and removes it from the internal configuration.

```
DWORD BSIF_DelSE (
    DWORD dwHandle,
    int iKEID,
    int iSEChn
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function stops all active jobs of the tightening channel (BSIF_Job_Abort) and deletes all allocated resources (BSIF_Job_Delete).

3.3 COMMUNICATION STATE

3.3.1 BSIF_GetKEState

This function returns information about the given communication channels current state.

```
DWORD BSIF_GetKEState (
    DWORD dwHandle,
    int iKEID,
    int *piState
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

piState
[out] Pointer to an integer-variable for holding the communications channel state (see below).

Return Values

A return value of zero indicates success. In this case the variable *piState contains a state value according to the following table. A nonzero return value indicates an error (see chapter 2.4 for details).
Allowed values for *piState:

Wert	Beschreibung
0	Invalid: not initialized
1	Idle: initialized, but not connected
2	Connecting: trying to start a connection
3	Connected: a connection is available and active
4	Paused: a communication channel has been paused temporarily by calling BSIF_Pause (used for serial connections to allow serial interface sharing if not in use by BS300IF-DLL, e. g. allow other processes to access the port (BS300 system configuration software))
5	Reconnect: a communication channel is going to be reconnected (e. g. after BSIF_Pause and BSIF_Continue).
6	Terminating: the communication channel is stopping (during shutdown)
7	Terminated: the communication has been terminated (shutdown)

Remarks

By calling this function one can check, if communication with the tightening system is active and running (e. g. if the TCP socket is connected). Note, that state information is supplied by the operating system, so this function might return “Connected” although no communication is actually possible (e. g. network plug removed). However, when trying to start a new job (send some data) state information is always updated (after operating system timeout).

3.4 JOBS, STATE AND DATA

3.4.1 BSIF_JobState

The function BSIF_JobState provides information on the current state of the given tightening channels current job.

```
DWORD BS300IF_JobState (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    int *piState,
    int *piResult
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

piState
[out] Pointer to an integer variable, receiving the current state after a successful execution of this function. The following values are possible (defined in bs300if_common.h):

Wert	Beschreibung
HEJS_Idle	Job not initialized or not active. This is the default state as long as the job is not active.
HEJS_Starting	The job has been started, but is not yet active.
HEJS_Started	The job has been started and is active.
HEJS_Running	The job is active and is waiting for completion (normally waiting for a response from the tightening system)
HEJS_Completing	Job has received a response from the tightening system and is now completing.
HEJS_Done (Flag)	If this flag is set, then the job is completed. The Job state is never HEJS_Done alone (this is a flag), but is used with the following completion state values:
HEJS_Done_OK	Job completed with success
HEJS_Done_ERR	Job completed with error
HEJS_Done_ETO	Job completed with timeout (Timeout-Error)
HEJS_Done_EAB	Job completed due to an abort (by calling BSIF_Job_Abort())

HEJS_Done_ENET	Job completed due to an network error
HEJS_Done_ES	Job completed due to an internal state error
HEJS_Done_WTO	Job completed with timeout warning (Timeout-Warning, only used with curve requests)

piResult

[out] Pointer to an integer variable, receiving an additional result value after successful call result value depends on object type and job state). The result value is not used at the moment by the BS300IF-DLL, but piResult must be non-NULL.

Return Values

A return value of zero indicates success (*piState and *piResult valid). Else an error code is returned (see chapter 2.4 for details).

Remarks

A successful job completion is indicated by the *piState value HEJS_Done_OK. A job always completes (at least due to timeout) by setting the flag HEJS_Done. The application logic can safely assume that a job after a successful call to BSIF_JobAdd_xxx completes after a short time and calls to the function BSIF_JobState return *piState with HEJS_Done set.

3.4.2 BSIF_JobDone

This function returns information if a job has been completed (with or without errors).

```
DWORD BS300IF_JobDone (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    BOOL *pbDone
);
```

Parameters

dwHandle

[in] Handle returned by calling BSIF_Init.

iKEID

[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn

[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

pbDone

[out] Pointer to a BOOL variable. After a successful call of this function its content will be set to TRUE, if the current job of the tightening channel has completed, else will be set to FALSE.

Return Values

A return value of zero indicates success (*pbDone valid). Else an error code is returned (see chapter 2.4 for details).

Remarks

This function works in the same way as BSIF_JobState, but decodes the internal state values in an easier to use way. See BSIF_JobState for more information on job completion states.

3.4.3 BSIF_JobSetTimeout

The function BSIF_JobSetTimeout provides a way of modifying the job timeout value.

```
DWORD BS300IF_JobState (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    DWORD dwMilliseconds
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

dwMilliseconds
[in] New job timeout value to use for this channel (in milliseconds). Do not set this value too low (below 100ms) or too high. The default value after starting communication is 10sec.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

After starting a job an internal timer gets started which supervises execution of the job. If the job does not finish before the timer times out, the job gets cancelled automatically and the jobs completion state is set to HEJS_DONETO (timeout). This mechanism guarantees, that each job started by the users application always completes.
Additional notes:

- If the timeout value is set too low, then the chance that a job might time out although a valid response is received from the SE/KE increases (due to network and internal thread synchronization delays) – so make sure to not set the timeout value too low (not below 100ms).
- Calling this function does not change the timeout value of a currently running job. The value specified here is used as a default value for the next time a new job is initialized/started.
- The timeout value of the 'KE'-job (this is relevant for all BSIF_JobAdd_xxx-functions which are called with an iKEID only (without an iSEChn parameter) may also be changed by supplying the special iSEChn value of 0x7F.

3.4.4 BSIF_Tools_CopyObjData

This functions copies job result data (payload) received from the tightening system into a user-provided buffer.

```
DWORD BSIF_Tools_CopyObjData (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    BYTE *pDstBuf,
    int *piDstLen
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

pDstBuf
[out] Pointer to the buffer that receives the result data (payload, bs-object) from the tightening controller. The buffer size must be greater or equal to the value provided by the parameter *piDstLen.

piDstLen
[in/out] Pointer to an integer variable. Before calling this function, the value should be initialized with the size of the buffer passed in pDstBuf. After a successful call, the value will be updated with the actual size of the job result data buffer.

Return Values

A return value of zero indicates success (*piDstLen and contents of piDstBuf valid). Else an error code is returned (see chapter 2.4 for details).

Remarks

This function should only be called after BSIF_JobState returns HEJS_Done_OK.
This function should be used instead of the obsolete function BSIF_JobResultData.

3.5 TIGHTENING CURVES

3.5.1 BSIF_JobAdd_BP_CURVE

This function sends a curve request command to the tightening system using default parameters.

```
DWORD BSIF_JobAdd_BP_CURVE (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    DWORD dwSeq
```

```
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

dwSeq
[in] Channels current sequence number for use with a curve request (0 for current/last).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a curve request command to the tightening system. The curve request mode used is publish/subscribe (see 2.3.1) and the default parameters used ask for a angle/torque curve. The response to this function is asynchronous, i. e. the response (and the curve data) will be delayed until the tightening system actually has a curve matching the request parameters. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.1.

3.5.2 BSIF_JobAdd_BP_CURVE_Ext

This function sends a curve request command to the tightening system and supplies additional parameters.

```
DWORD BSIF_JobAdd_BP_CURVE_Ext (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    BSIF_CurveParams* pParams
);
```

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

pParams
[in] Pointer to a structure of type BSIFCurveParams with variable size (see below).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a curve request command to the tightening system. The curve request mode can be defined by setting the corresponding parameters inside pParams (see below). The response to this function is asynchronous, i. e. the response (and the curve data) will be delayed until the tightening system responds with a curve matching the request parameters. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.
The data structure BSIFCurveParams is a structure with variable size and has the following elements:

Element	Description
dwLength	Total size of this structure (in bytes)
dwVersion	Version number of this structure (must be zero for now)
bMode	Curve request mode (a value from the enum eCPMode, siehe bs300if_curve.h): ecpmOnline — Online request of cure data (Publish/Subscribe) ecpmBuffer — Curve buffer request
bAction	Curve action (Bitmask, use a combination of values from enum eCPAction), siehe bs300if_curve.h): ecpaSubscribe — Activate request (normally this should be set) ecpaUnsubscribe — Abort current request (not used) ecpaFilterPrg — Activate tightening program filters ecpaFilterQC — Activate quality code filter
bType	Curve type (Bitmask, use a combination of a maximum of three values from enum eCPType) ecptAngle — Angle ecptTorque — Torque ecptGradient — Gradient ecptTime — Time ecptTRed — Redundancy torque
bMask	Result filter mask (Bitmask, use a combination of 0x01=OK and 0x02=NOK, e. g. 0x03 requests all curves regardless of OK/NOK)
dwSeq	Sequence number of requested curve (0 for last/current curve)

3.5.3 BSIF_Crv_GetCrvData

This function analyses and decodes a curve object and copies curve data into an easy to use data structure.

```
DWORD BSIF_Crv_GetCrvData (
    BYTE *pSrcBuf,
    int iSrcLen,
    BSIFCurveData *pData
);
```

Parameters

pSrcBuf
[in] Pointer to the curve object returned from BSIF_Tools_CopyObjData after a curve request has completed successfully (e. g. after calling BSIF_JobAdd_BP_CURVE or BSIF_JobAdd_BP_CURVE_Ext).

iSrcLen
[in] Integer-Value with the size of the buffer supplied in pSrcBuf (object length, returned from BSIF_Tools_CopyObjData).

pData

[in/out] Pointer to a variable length structure of type BSIFCurveData. Before calling this function the member dwSize must be initialized with the available buffer size in bytes. After a successful call pData will contain the decoded curve data.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

The data structure BSIFCurveData is a variable length data structure and has the following elements:

Element	Description
dwSize	Total size of this structure (in Bytes)
Status	Status of curve transmission (one value from the enumeration eCDStatus, see bs300if_curve.h): ecdsOk — curve data OK ecdsEBufEmpty — Data buffer empty ecdsEReqInvType — Request for invalid curve type ecdsEWait — Waiting for curve ecdsEInvBuf — Wrong memory ecdsENoZip — Data compression not possible ecdsEInvAct — invalid action ecdsEInvVer — Wrong curve version requested ecdsEReSend — curve already sent ecdsEUnsub — Correct curve unsubscribe ecdsENoUnsub — curve unsubscribe unnecessary ecdsEOOR — Max. BS accesses exceeded ecdsENoRed — No redundancy transducer activated ecdsENoFilt — Filter parameters not fulfilled ecdsECfgChanged — Configuration changed ecdsESEStopped — Curve storage in SE interrupted ecdsESEDeleted — Curve deleted on SE ecdsECfgDeleted — Configuration deleted ecdsECrvDeleted — Curve deleted ecdsECfgError — Configuration incorrect ecdsEMemFull — Linear memory full ecdsECRC — CRC error ecdsEInvSequence — Request for incorrect sequence number ecdsEUsrStop — Curve storage stopped by other user ecdsEUsrStart — Curve storage started by another user ecdsENoProg — Curve storage stopped, because no program is selected
DateTime	Date/Time. This is a variant of VT_DATE.
Sequence	Channel sequence counter value (cycle counter)
IDCode	ID-Code
CurveType	Curve type (Bitmask, combination of max. three values of eCPType, see bs300if_curve.h) ecptAngle — Angle ecptTorque — Torque ecptGradient — Gradient ecptTime — Time ecptTRed — Redundancy-Torque
CurveOrds	Number of bits in CurveType (number of axes)
Dimension	Dimension of the values (one value from enum eCDDim, see bs300if_curve.h):

	ecddNm Nm ecddFtLb Ftlb ecddInlb Inlb ecddKpm Kpm
OkNok	Tightening result (1 = OK, 0/-1 = NOK)
QC1, QC2	Quality code
Program	Program number
Channel	Channel number (in RS-format)
ReworkCode	Rework code
TTAct	Torque threshold actual value of last step
T_Min, T_Max, T_Act	Torque actual value and limits of last step
A_Min, A_Max, A_Act	Angle actual value and limits of last step
G_Min, G_Max, G_Act	Gradient actual value and limits of last step
TmMin, TmMax, TmAct	Time: actual value and limits of last step
Direction	Direction of last step: 1 = CW (clockwise), 0 = CCW (counterclockwise)
Row, Column	Row- and column-number of last step (0…n)
TTIdx	Curve point index for start of angle counting in the last step (torque threshold index). This value is < 0 if the point is not within the available curve data area.
StepCount	Number of steps
MonitorCount	Number of monitoring functions
TargetCount	Number of target functions
PointCount	Number of curve points
steps	Pointer to list of steps (elements are of type tCDStepItem)
monitors	Pointer to list of monitoring function values (elements are of type tCDMFItem)
targets	Pointer to list of target function values (elements are of type tCDTFItem)
points	List of curve points. The memory layout is as follows (the order is as the order of the dimensions in eCPType): [0] → Point [0], Dimension [0] [1] → Point [0], Dimension [1] [2] → Point [1], Dimension [0] [3] → Point [1], Dimension [1] (The same for curves with tripels (three axes/dimensions): 0…2→Point [0], 3…5 → Point 1, …)
Data	Buffer into which the pointers steps, monitors, targets and points point into.

Within this structure there are different arrays containing information about the curve data elements. Tightening steps are described in the table steps. Each item of this table has the following structure (type tCDStepItem):

Element	Description
Row	Steps row number(0 ⇔ Zeile 1)
Col	Steps column number (0 ⇔ Spalte ‚A')
EndIdx	Curve point index to locate the end of the step within the curve points. If the point is not within the points array, then the value is < 0 (may happen due to the limited curve buffer size of max. 2000 curve points).

The monitoring function values are listed in the table monitors. Each item has the following structure (type tCDMFItem):

Element	Description
ID	Unique ID of the monitoring function. ID's for monitoring functions and target function are listed in the enumeration eCDFNCode (see below)

Minimum	Lower monitoring limit, coded as VARIANT. VarType might be either R4 (float) or I4 (Long) or even empty.
Maximum	Lower monitoring limit, coded as VARIANT. VarType might be either R4 (float) or I4 (Long) or even empty.
Actual	Actual measured value, coded as VARIANT. VarType might be either R4 (float) or I4 (Long) or even empty.

The target function values are listed in the table targets. Each item has the following structure (type tCDTFItem):

Element	Description
ID	Unique ID of the target function. ID's for monitoring functions and target function are listed in the enumeration eCDFNCode (see below)
Target	Target value as defined in tightening program (set value), coded as VARIANT. VarType might be either R4 (float) or I4 (Long) or even empty.
Actual	Actual measured value, coded as VARIANT. VarType might be either R4 (float) or I4 (Long) or even empty.

All ID's for target functions and monitoring functions are shown in the following table (values are defined in the enumeration eCDFNCode, defined in bs300if_curve.h). The columns „M" and „T" show, if the value is allowed as monitoring function („M") and/or target function („T").

Element	M	T	Description
ecdfnTorque	M	T	Torque
ecdfnAngle	M	T	Angle
ecdfnTime	M	T	Time
ecdfnInput		T	Input signal
ecdfnYieldPoint		T	Yield Point
ecdfnSync		T	Synchronisation
ecdfnTUpper		T	target upper torque MP
ecdfnTUpperRel		T	target upper torque relative MP&
ecdfnTLower		T	target lower torque MPu
ecdfnTLowerRel		T	target lower torque relative MP&u
ecdfnGradient	M	T	Gradient GP
ecdfnTorqueEx	M		extended torque
ecdfnTorqueRel	M		torque relative M&
ecdfnTorqueMax	M		torque maximum M>
ecdfnTorqueMin	M		torque minimum M<
ecdfnTorqueAvg	M		torque average MM
ecdfnTorqueRange	M		torque range M[
ecdfnAngleTotal	M		total angle WG
ecdfnAbove	M		tightening from above MEo
ecdfnBelow	M		tightening from below MEu
ecdfnGradAvg	M		gradient average GM
ecdfnUnused			empty/unused
ecdfnUnknown			unknown/invalid

3.5.4 BSIF_Crv_Save

This function saves a curve object into a .crv file to be read by the System configuration tool BS300.

```
DWORD BSIF_Crv_SaveCrv (
    const char *pszFilename,
    BYTE *pbBuf,
```

```
    int iLen
);
```

Parameters

pszFilename
[in] Pointer to an ASCIIZ-string. The string is used as filename for the curve file to be written.

pbBuf
[in] Pointer to a curve object (e. g. received from the tightening system by calling BSIF_JobAdd_BP_CURVE_Ext and BSIF_Tools_CopyObjData).

iLen
[in] Size of the object passed in pbBuf.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

The buffer must contain a curve object. The file generated is compatible with the native .crv file format used in BS300.

3.6 TIGHTENING PROGRAMS AND APPLICATIONS

3.6.1 BSIF_JobAdd_BP_READ_SRBPRG

This function starts a tightening program download request to transfer a tightening program from the tightening controller to the PC.

```
DWORD BSIF_JobAdd_BP_READ_SRBPRG (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    int iProgram
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

iProgram
[in] Parameter set number (program number) of the tightening program to load (0…47 and 99 for firmware versions <= 2.000).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a tightening program download request command to the tightening system. The response to this function is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.

3.6.2 BSIF_JobAdd_BP_WRITE_SRBPRG

This function uploads a tightening program from the PC to the tightening controller.

```
DWORD BSIF_JobAdd_BP_WRITE_SRBPRG (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    int iDstProg,
    int iSrcProg,
    BYTE *pbSrcBuf,
    int iSrcLen
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

iDstPrg
[in] Parameter set number (program number) of the destination tightening program inside the tightening controller (0…47 and 99 for firmware versions <= 2.000).

iSrcPrg
[in] If this function is called with a complete .prx program file in pbSrcBuf, then this parameter can be used to extract a single tightening program from the file buffer (0…47 und 99). If there is a single tightening program object in pbSrcBuf (e. g. downloaded using BSIF_JobAdd_BP_READ_SRBPRG and BSIF_Tools_GetObjData), then a value of -1 must be used.

pbSrcBuf
[in] Pointer to memory block containing either a single tightening program object (then iSrcPrg must be equal to -1) or the complete content of a .prx file.

iSrcLen
[in] Size of the buffer (in bytes) provided in pbSrcBuf.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a tightening program upload request command to the tightening system. The response to this function (and therefore the information, if the upload was successful and the program contents are valid) is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.
Note, that only after a response from the tightening controller has been received, one can check if the tightening program was transmitted and stored successfully (the controller runs additional checks on the tightening program after it was received).

Warning

When uploading a tightening program to the controller one should double-check, if the tightening program was actually correctly uploaded (e. g. by additionally downloading and comparing again)!
Note also, that a successful program upload does not necessarily mean the tightening program will actually run. This is not due to a problem in the BS300IF, but due to additional validity checks performed by the tightening controller during uploading and executing of the tightening program. E. g. the controller checks for the maximum allowed speed or maximum allowed torque for the tool connected to the controller.

3.6.3 BSIF_JobAdd_BP_DEL_ALL_PRG

This function deletes all tightening programs currently stored inside a single controller (SE/CS).

```
DWORD BSIF_JobAdd_BP_DEL_ALL_PRG (
    DWORD dwHandle,
    int iKEID,
    int iSEChn
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a “delete all tightening programs” command to a tightening channels controller. Using this functions can speed up uploading tightening programs from the PC to the controller significantly in case of multiple uploads (“restore” scenario). Normally uploading a program means erasing one or more flash memory sectors inside the controller (this takes long time), then writing it to the flash memory. If the whole tightening program flash memory gets erased before uploading, then single erases are not necessary for further uploads, therefore speeding up the whole process.

The response to this function (and therefore the information, if the command was successful executed) is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.

3.6.4 BSIF_Prg_ReadPrg

This function reads a tightening program definition file (*.prg) and copies the tightening object data into a user supplied buffer.

```
DWORD BSIF_Prg_ReadPrg (
    const char *pszFilename,
    BYTE *pbBuf,
    int *piLen
);
```

Parameters

pszFilename

[in] Pointer to an ASCIIZ-string. The string is used as filename for the tightening program file to be read (*.prg).

pbBuf

[in/out] Pointer to a buffer to receive (after successful execution of the function) the tightening program object. The data returned in the buffer can be used in a call BSIF_Prg_GetPrgData to decode the tightening program.

piLen

[in/out] Pointer to a variable which (on input) contains the size of the user buffer. On output returns the actual size of the tightening program object read from the given file.

Return Values

A return value of zero indicates success (*piLen and pbBuf valid). Else an error code is returned (see chapter 2.4 for details).

Remarks

On successful execution, the result buffer contains a tightening program object which might be used to call BSIF_Prg_GetPrgData for decoding.

3.6.5 BSIF_Prg_ReadPrx

This program read a tightening program object from a tightening program compound file (*.prx). It extracts the tightening program object by the given program number and copies it into a user supplied buffer.

```
DWORD BSIF_Prg_ReadPrx (
    const char *pszFilename,
    int iPrg,
    BYTE *pbBuf,
    int *piLen
);
```

Parameters

pszFilename
[in] Pointer to an ASCIIZ-string. The string is used as filename for the tightening program file to be read (*.prx).

iPrg
[in] Program number (parameter set number) of the tightening program object to extract from the file.

pbBuf
[in/out] Pointer to a buffer to receive (after successful execution of the function) the tightening program object. The data returned in the buffer can be used in a call BSIF_Prg_GetPrgData to decode the tightening program.

piLen
[in/out] Pointer to a variable which (on input) contains the size of the user buffer. On output returns the actual size of the tightening program object read from the given file.

Return Values

A return value of zero indicates success (*piLen and pbBuf valid). Else an error code is returned (see chapter 2.4 for details).

Remarks

On successful execution, the result buffer contains a tightening program object which might be used to call BSIF_Prg_GetPrgData for decoding.

3.6.6 BSIF_Prg_SavePrg

This function saves a tightening program object into a file (*.prg). It uses the same file format as BS300, so a file saved using this function can be edited/viewed using BS300.

```
DWORD BSIF_Prg_SavePrg (
    const char *pszFilename,
    BYTE *pbBuf,
    int iLen
);
```

Parameters

pszFilename
[in] Pointer to an ASCIIZ-string. The string is used as filename for the tightening program file to be saved (*.prg).

pbBuf
[in] Pointer to a buffer which contains the tightening program object to be written to a file (e. g. data returned from BSIF_JobAdd_BP_READ_SRBPRG/BSIF_CopyObjData).

iLen
[in] Length of buffer used in pbBuf (size of the tightening program object data).

Return Values

A return value of zero indicates success file saved). Else an error code is returned (see chapter 2.4 for details).

Remarks

The buffer must contain a tightening program object. The file is written is compatible to the native BS300 *.prg file format.

3.6.7 BSIF_Prg_GetPrgData

This function analyses and decodes a tightening program object and copies it data into an easy to use data structure.

```
DWORD BSIF_Prg_GetPrgData (
    BYTE *pSrcBuf,
    int iSrcLen,
    BSIFPrgData *pData
);
```

Parameters

pSrcBuf

[in] Pointer to a tightening program object (e. g. received by calling on of the functions BSIF_Prg_ReadPrg or BSIF_JobAdd_BP_READ_SRBPRG/BSIF_Tools_CopyObjData).

iSrcLen

[in] Integer-Value with the size of the buffer supplied in pSrcBuf (object length, returned from BSIF_Tools_CopyObjData).

pData

[in/out] Pointer to a variable length structure of type BSIFPrgData. Before calling this function the member dwSize must be initialized with the available buffer size in bytes. After a successful call pData will contain the decoded tightening program object data.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

The data structure BSIFPrgData is a variable length data structure and has the following elements:

Element	Beschreibung
dwSize	Total size of the structure (in Bytes). On input: available user buffer size (in Bytes).
Dimension	Dimesion used (0 = Nm, 1 = FtLb, 2 = InLb, 3 = Kpm)
DateTime	Last change or file write modification date/time of tightening program object. This is a variant of VT_DATE.
Name	Name of the tightening program (ASCIIZ)
Comment	Optional tightening program comment (ASCIIZ)
StepCount	Number of elements in table steps
ParamCount	Number of elements in table params
steps	Pointer to a table (array) with elements of type tBSIFPDStepItem (containing information about the tightening steps of this program). The pointer always points to an address located within this structure (after the fixed area).
params	Pointer to a table (array) with elements of type tBSIFPDParamItem (containing information about the tightening parameters and their values. The pointer always points to an address located within this structure (after the fixed area).

Data	Variable length buffer containing step- and parameter- tables.

Within this structure there are different tables/arrays containing information about the tightening program. Tightening steps are described in the table steps. Each item of this table has the following structure (type tBSIFPDStepItem):

Element	Beschreibung
Row	Steps row number (0 ⇔ row 1)
Col	Steps column number (0 ⇔ column ‚A')
Typ	Type of this tightening step. Type contains one value of the enumeration eBSIFParStep (see bs300if_program.h): ebspstStart — Start step ebspstTighten — Standard tightening step ebspstExtTighten — Extended tightening step ebspstJump — Step containing a jump to another step ebspstSync — Sync-step (for application synchronization) ebspstOkNokBranch — OK/NOK-branch ebspstBranchInput — Branch step for input signals ebspstRework — Rework step (application rework) ebspstEnd — End step
Name	Name of the tightening step (ASCIIZ)

The actual tightening program parameters are contained in the table params. Each item of this table has the following structure (type tBSIFPDParamItem):

Element	Beschreibung
Row	Steps row number (0 ⇔ row 1)
Col	Steps column number (0 ⇔ column ‚A')
ID	Unique ID of the parameter
Code	Type-code of the parameter. The type code alone does not fully define the parameter. To fully define the parameter a combination of the fields "Code", "Flags" and "Dim" must be used (see sample code). Code is one of the values defined in the enumeration eBSIFParCode (see bs300if_program.h): ebspcoTarget — Target function value (e. g. target function torque) ebspcoMin — Minimum value (e. g. Angle monitoring minimum) ebspcoMax — Maximum value ebspcoTHTorque — Threshold torque to start tightening measurements ebspcoTHAngle — Threshold torque to start angle counting ebspcoTHGradient — Threshold torque to start gradient measurements ebspcoYieldPoint — Yield point [%] ebspcoAngleCheck — Angle check value ebspcoInput — Input signal ebspcoAngleFactor — Angle correction factor ebspcoTorqueFactor — Torque correction factor ebspcoAngleFilter — Angle filter factor ebspcoChordAngle — Chord angle ebspcoTolerance — Tolerance value, e. g. allowed red/act difference ebspcoErgoStart — ErgoStart-Value ebspcoErgoStop — ErgoStop-Value ebspcoStartupSupp — Startup suppression value ebspcoSync — Synchronization ebspcoNOKAbort — NOK-Abort ebspcoBoltSelect — Bolt selection ebspcoRework — Rework

	ebspcoRuns	Runs
	ebspcoInpCount	Input count
	ebspcoReworkTab	Rework table
	ebspcoDrawing	Drawing value
	ebspcoRelTorque	relative torque monitoring
	ebspcoTHRef	Threshold torque reference, abs/rel
	ebspcoTriggerOut	Trigger output
	ebspcoTFLow	Target function low (TF)
	ebspcoMinAvg	Minimum average
	ebspcoMaxAvg	Maximum average
	ebspcoMinRange	Minimum range
	ebspcoMaxRange	Maximum range
	ebspcoMinTotal	Minimum Total (global)
	ebspcoMaxTotal	Maximum Total (global)
	ebspcoSSbelow	Count of Stick Slip below
	ebspcoTHSS	Stick Slip threshold torque
	ebspcoMinTorque	Minimum “tighteining from” (above/below)
	ebspcoMaxTorque	Maximum “tighteining from” (above/below)
	ebspcoLBAngleL	Lookback angle below
	ebspcoLBAngleU	Lookback angle above
	ebspcoMAngleL	Monitoriung angle below
	ebspcoMAngleU	Monitoring angle above
	ebspcoDocuStep	Dokubuffer
	ebspcoQCCategory	Quality code category
	ebspcoDataSupp	Data output suppressed
	ebspcoTHYieldPoint	Threshold for yield point
	ebspcoRPM	Rotion speed
	ebspcoCurveResol	Curve resolution
	ebspcoCurvePoints	Number of curve points
	ebspcoGradAvg	Gardient average
	ebspcoAvgInterval	Intervall size for gradient averaging
	ebspcoUEFMax	Minimum UEF Torque M>+
	ebspcoUEFMin	Maximum UEF Torque M
Flags	The value of Flags supplies additional information for a parameter. Flags is a combination of the values of the enumeration eBSIFParFlags:	
	ebspafRedundancy	Parameter refers to a redundancy measurement
	ebspafRelative	Parameter is a relative value
Dim	The value of Dim supplied additional information for a parameter. Flag is one of the value defined in the enumeration eBSIFParDim:	
	ebspadNoDim	Parameter has no dimension
	ebspadTorque	Parameter is a torque value [Nm, FtLb, ...]
	ebspadAngle	Parameter is an angle value
	ebspadTime	Parameter is a time value [s]
	ebspadPercentage	Parameter is a percentage [%]
	ebspadGradient	Parameter is a gradient value [Nm/s, FtLb/s, ...]
	ebspadRPM	Parameter is a rotation speed value [RPM]
	ebspadValue	Parameter is a value/number
	ebspadFactor	Parameter is a factor
	ebspadNumber	Parameter is a count
	ebspadFlag	Parameter is a flag
	ebspadChannel	Parameter is a channel mapping
Val	The actual parameter value, as VARIANT. VarType is either R4 (float) or I4 (long).	

To decode a parameter the combination of Code, Flags und Dim must be used. Examples:

ebspcoAngleFactor + ebspadFlag → Angle correction flag: aktiv/inaktiv

ebspcoAngleFactor + ebspadFactor	→ Angle correction factor: correction factor
ebspcoMax + ebspadTorque	→ Monitoring value, max. torque
ebspcoMax + ebspadTorque + ebspafRedundancy	→ Monitoring value, max. redundancy torque

3.6.8 BSIF_Prg_SetParam

This function modifies a parameter value in a tightening program object.

```
DWORD BSIF_Prg_SetParam (
    BYTE *pSrcBuf,
    int iSrcLen,
    int iRow,
    int iCol,
    int iParamID,
    float fValue
);
```

Parameters

pSrcBuf
[in/out] Pointer to the tightening program object data buffer which should be modified. (e. g. aquired by BSIF_Prg_ReadPrg or BSIF_JobAdd_BP_READ_SRBPRG/BSIF_Tools_CopyObjData).

iSrcLen
[in] Integer-value containing the size of the buffer referenced through pSrcBuf (size of the tightening program object).

iRow
[in] Integer-Value identifying the row of the tightening step to modify (Steps row number (0 ⇔ row 1), e. g. step “4F” → iRow = 3).

iCol
[in] Integer-Value identifying the column of the tightening step to modify (Steps column number (0 ⇔ row ‘A’), e. g. step “4F” → iCol = 5).

iParamID
[in] Integer-value identifying the parameter set to be changed. The value must be one of the iParamID values from the struct params (Typ: tBSIFPDParamItem) previously read by BSIF_Prg_GetData.

fValue
[in] float value of the new parameter value.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

By calling this function the tightening program object is modified. The given parameter value is changed and checksums of the tightening program object are updated.

Warning

Updating a tightening program object might create a tightening program object which cannot be uploaded to a channel or could lead to an error when trying to run the tool using this parameter set. This is not due to a problem in the BS300IF, but due to additional validity checks performed by the tightening controller during uploading and executing of the tightening program. E. g. the controller checks for the maximum

allowed speed or maximum allowed torque for the tool connected to the controller. Uploading a tightening program which exceeds the allowed specifications can therefore lead to an error.
When uploading a modified program you should also double check your modifications to the tightening program work as expected. We recommend to re-download the tightening program after uploading it and to compare it to ensure the upload succeeded as expected. We also recommend to at least use the data output of the controller (tightening result data) and compare the actual tightening result values with the tightening program objects parameter values. This gives an additional consistency check between expected parameters/result values and actual process values. Any inconsistency must be seriously checked and reported!

3.6.9 BSIF_JobAdd_BP_READ_SRBAPP

This function starts a tightening application download request to transfer a tightening application definition from the tightening cell controller (KE) to the PC.

```
DWORD BSIF_JobAdd_BP_READ_SRBAPP (
    DWORD dwHandle,
    int iKEID,
    int iAppSeq
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iAppSeq
[in] Application sequence number (application number) of the tightening application to load (0…47 for firmware versions <= 2.000).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a tightening application download request command to the tightening system. The response to this function (and therefore the information, if the command was successful executed) is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.

3.6.10 BSIF_JobAdd_BP_WRITE_SRBPRG

This function uploads a tightening application from the PC to the tightening cells controller (KE).

```
DWORD BSIF_JobAdd_BP_WRITE_SRBPRG (
    DWORD dwHandle,
    int iKEID,
    int iAppSeq,
    BYTE *pbSrcBuf,
```

```
    int iSrcLen
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iAppSeq
[in] Application sequence number to use for storing this application inside the tightening cell controller (KE), i. e. “target application sequence number”.

pbSrcBuf
[in] Pointer to memory block containing a single tightening application object.

iSrcLen
[in] Size of the buffer (in bytes) provided in pbSrcBuf.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a tightening application upload request command to the tightening system. The response to this function (and therefore the information, if the upload was successful and the program contents are valid) is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.
Note, that only after a response from the tightening controller has been received, one can check if the tightening application was transmitted and stored successfully (the controller runs additional checks on the tightening application after it was received).

Warning

When uploading a tightening application to the controller one should double-check, if the tightening application was actually correctly uploaded (e. g. by additionally downloading and comparing again)!
Note also, that a successful upload does not necessarily mean the tightening application will actually run. This is not due to a problem in the BS300IF, but due to additional validity checks performed by the tightening controller during uploading and executing of the tightening application. E. g. the controller checks if the configured tools are really online.

3.6.11 BSIF_JobAdd_BP_DEL_ALL_APP

This function deletes all tightening applications currently stored inside the tightening cell controller (KE).

```
DWORD BSIF_JobAdd_BP_DEL_ALL_APP (
    DWORD dwHandle,
    int iKEID
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a “delete all tightening applications” command to a tightening cells controller (KE). Using this functions can speed up uploading multiple tightening applications from the PC to the controller significantly. Normally uploading an application means erasing one or more flash memory sectors inside the controller (this takes long time), then writing it to the flash memory. If the whole tightening application flash memory gets erased before uploading, then single erases are not necessary for further uploads, therefore speeding up the whole process.
The response to this function (and therefore the information, if the command was successful executed) is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.

3.7 TIGHTENING SYSTEM INFORMATION

3.7.1 BSIF_JobAdd_BP_LOGONREQ

This function starts a logon request to get the list of active tightening controller modules and their properties (channel numbers, name & version, physical location).

```
DWORD BSIF_JobAdd_BP_LOGONREQ (
    DWORD dwHandle,
    int iKEID
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE. Note, that it is possible to use this command regardless if connected physically to a KE or a SE/CE. This function sends a broadcast to the given communication channel (given by KEID) and returns result information according to the actual physical connection.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a logon request command to the tightening system. Note, that it is possible to use this command regardless if physically connected to a KE or a SE/CE. This function sends a broadcast to the given communication channel (given by KEID) and returns result information according to the actual physical connection.
The response to this function is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3. After the response is received, it can be decoded using the functions BSIF_Tools_CopyObjData and BSIF_Sys_GetLogonData (see chapter 3.7.2).

3.7.2 BSIF_Sys_GetLogonData

This function analyses and decodes a logon response object and copies data into an easy to use data structure.

```
DWORD BSIF_Sys_GetLogonData (
    BYTE *pSrcBuf,
    int iSrcLen,
    BSIFSysLogonData *pData
);
```

Parameters

pSrcBuf
[in] Pointer to the curve object returned from BSIF_Tools_CopyObjData after a curve request has completed successfully (e. g. after calling BSIF_JobAdd_BP_CURVE or BSIF_JobAdd_BP_CURVE_Ext).

iSrcLen
[in] Integer-Value with the size of the buffer supplied in pSrcBuf (object length, returned from BSIF_Tools_CopyObjData).

pData
[in/out] Pointer to a variable length structure of type BSIFSysLogonData. Before calling this function the member dwSize must be initialized with the available buffer size in bytes. After a successful call pData will contain the decoded logon data.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function decodes the logon data response from the tightening unit. The resulting structure contains information about the physical setup of the tightening system, its modules firmware versions and their names. For each module plugged into the tightening system the structure supplies the rack- and slot-number where each module is located in the system.
The data structure BSIFSysLogonData is a variable length data structure and has the following elements:

Element	Description
dwSize	Total size of this structure (in Bytes)
ChnCnt	Number of channel information items in tBSIFChnLogonData array (ChnInfo).
CURackSlot	Physical location of communication partner. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

KEIndex	Index of KE in tBSIFChnLogonData array. If no KE available, then -1.
ChnInfo	Pointer to array of channel info items (elements are of type tBSIFChnLogonData)
data	Buffer into which the pointer ChnInfo points into.

Channel info items are described in the array ChnInfo. Each item of this array has the following structure (type tBSIFChnLogonData):

Element	Description
Channel	Rack/Slot of this module
Version	This modules version string (ASCII-Z)
Name	This modules name as defined by BS300/BS350 Administration → Location Names (ASCII-Z)

3.7.3 BSIF_JobAdd_BP_SYS_APPNAMTAB

This function starts a request to get the tightening application table from a KE. The tightening application table provides a list of configured tightening applications and their name.

```
DWORD BSIF_JobAdd_BP_SYS_APPNAMTAB (
    DWORD dwHandle,
    int iKEID
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a request command to get the application table from the tightening system cell controller (KE).
The response to this function is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3. After the response is received, it can be decoded using the functions BSIF_Tools_CopyObjData and BSIF_Sys_GetAppNameTbl (see chapter 3.7.4).

3.7.4 BSIF_Sys_GetAppNameTbl

This function analyses and decodes a tightening application table response object returned after executing BSIF_JobAdd_BP_SYS_APPNAMTAB and copies data into an easy to use data structure.

```
DWORD BSIF_Sys_GetAppNameTbl (
    BYTE *pSrcBuf,
    int iSrcLen,
```

```
    BSIFSysNameTable *pData
);
```

Parameters

pSrcBuf

[in] Pointer to the curve object returned from BSIF_Tools_CopyObjData after a application table request has completed successfully (e. g. after calling BSIF_JobAdd_BP_SYS_APPNAMTAB).

iSrcLen

[in] Integer-Value with the size of the buffer supplied in pSrcBuf (object length, returned from BSIF_Tools_CopyObjData).

pData

[in/out] Pointer to a variable length structure of type BSIFSysNameTable. Before calling this function the member dwSize must be initialized with the available buffer size in bytes. After a successful call pData will contain the decoded application table data.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function decodes the application table response from the tightening cell controller (KE). The resulting structure contains a list of application numbers and application names currently defined inside the KE. The data structure is used for the list of tightening programs inside a SE/CS and the list of applications inside the KE.
The data structure BSIFNameTable is a variable length data structure and has the following elements:

Element	Description
dwSize	Total size of this structure (in Bytes)
ItemCnt	Number of items in *tBSIFNamedItem* array (Items).
Items	Pointer to array of named items (elements are of type *tBSIFNamedItem*)
Data	Buffer into which the pointer *Items* points into.

Named info items are described in the array *Items*. Each item of this array has the following structure (type *tBSIFNamedItem*):

Element	Description
Num	Item number (i. e. tightening application number “sequence number”)
Name	This items name (i. e. tightening application name, ASCII-Z)

3.7.5 BSIF_JobAdd_BP_SYS_PRGNAMTAB

This function starts a request to get the tightening program table from a SE/CS. The tightening program table provides a list of configured tightening programs and their name for a given tightening channel.

```
DWORD BSIF_JobAdd_BP_SYS_PRGNAMTAB (
    DWORD dwHandle,
    int iKEID,
    int iSEChn
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a request command to get the list of configured tightening programs and their names (tightening program table) from a tightening channel (SE/CS).
The response to this function is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3. After the response is received, it can be decoded using the functions BSIF_Tools_CopyObjData and BSIF_Sys_GetPrgNameTbl (see chapter 3.7.6).

3.7.6 BSIF_Sys_GetPrgNameTbl

This function analyses and decodes a tightening application table response object returned after executing BSIF_JobAdd_BP_SYS_PRGNAMTAB and copies data into an easy to use data structure.

```
DWORD BSIF_Sys_GetPrgNameTbl (
    BYTE *pSrcBuf,
    int iSrcLen,
    BSIFSysNameTable *pData
);
```

Parameters

pSrcBuf
[in] Pointer to the curve object returned from BSIF_Tools_CopyObjData after a application table request has completed successfully (e. g. after calling BSIF_JobAdd_BP_SYS_PRGNAMTAB).

iSrcLen
[in] Integer-Value with the size of the buffer supplied in pSrcBuf (object length, returned from BSIF_Tools_CopyObjData).

pData
[in/out] Pointer to a variable length structure of type BSIFSysNameTable. Before calling this function the member dwSize must be initialized with the available buffer size in bytes. After a successful call pData will contain the decoded application table data.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function decodes the tightening program table response from a tightening channel (SE/CS). The resulting structure contains a list of tightening program numbers and their names currently defined inside the tightening channel. The data structure is used for the list of tightening programs inside a SE/CS and the list of applications inside the KE.
The data structure BSIFNameTable is a variable length data structure and has the following elements:

Element	Description
dwSize	Total size of this structure (in Bytes)
ItemCnt	Number of items in *tBSIFNamedItem* array (Items).
Items	Pointer to array of named items (elements are of type *tBSIFNamedItem*)
Data	Buffer into which the pointer *Items* points into.

Named info items are described in the array *Items*. Each item of this array has the following structure (type *tBSIFNamedItem*):

Element	Description
Num	Item number (i. e. tightening program number)
Name	This items name (i. e. name of tightening program, ASCII-Z)

3.7.7 BSIF_JobAdd_BP_Diag

This function sends a diagnostics request command to the tightening system and supplies additional parameters.

```
DWORD BSIF_JobAdd_BP_Diag (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    BSIFDiagParams* pParams
);
```

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2). To query the tightening cell controller (KE) supply a value of 0x7F.

pParams
[in] Pointer to a structure of type BSIFCurveParams with variable size (see below).

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends a diagnostics request command to the tightening system. The diagnostics request type can be defined by setting the corresponding parameters inside pParams (see below). The response to this function is asynchronous, i. e. the response (and the diagnostics result data) will be delayed until the

tightening system responds with diagnostics data. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.
The data structure BSIFDiagParams is a structure with variable size and has the following elements for dwLevel of zero:

Element	Description
dwSize	Total size of this structure (in bytes)
dwLevel	Information level of this structure (must be zero at the moment)
bComponent	Diagnostics component (one of the following values): 0x00 Request SE3xx controller information (System 300/310 only) 0x02 Request MC1 spindle transducer information (not for ErgoSpin) 0x0F Request ErgoSpin tool information (ErgoSpin only) 0x16 Request CS351 controller information (System 350 only)
bAction	Diagnostics action (one of the following values): 0x00 Read diagnostics information

Note that getting diagnostics information from the tightening system is usually a multi-step process. For example to get the tool serial numbers, you must walk the “hardware chain” as follows:

1. Use BSIF_JobAdd_BP_LOGONREQ and BSIF_Sys_GetLogonData to get the module type of a given tightening controller.
2. Depending on the module type (CS351 or SExxx) execute a BSIF_JobAdd_BP_Diag command to get information about the controller module (e. g. controller serial number, firmware version, …) and the type of the connected tool (handtool ErgoSpin or tightening spindle). Use bComponent 0x16 for CS351 and 0x00 for SExxx.
3. Depending on the type of the connected tool execute another BSIF_JobAdd_BP_Diag command to get information about the tool (e. g. tool serial number). Use bComponent 0x0F for hand tools and 0x02 for spindles.

3.7.8 BSIF_Sys_GetDiagData

This function analyses and decodes a diagnostics object and copies diagnostics data into an easy to use data structure.

```
DWORD BSIF_Sys_GetDiagData (
    BYTE *pObjBuf,
    int iObjLen,
    BSIFDiagData *pData
);
```

Parameters

pObjBuf
[in] Pointer to the curve object returned from BSIF_Tools_CopyObjData after a diagnostics request has completed successfully (e. g. after calling BSIF_JobAdd_BP_Diag).

iobjLen
[in] Integer-Value with the size of the buffer supplied in pSrcBuf (object length, returned from BSIF_Tools_CopyObjData).

pData
[in/out] Pointer to a variable length structure of type BSIFDiagData. Before calling this function the member dwSize must be initialized with the available buffer size in bytes. Also the member dwLevel

must be initialized with one of the allowed information levels. After a successful call pData will contain the decoded curve data.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details). Please note, that the function also returns an error (HE_E_INVALIDOBJ), if the requested information level (pData->dwLevel) does not match the object (pObjBuf) passed to the function.

Remarks

The data structure *BSIFDiagData* is a variable length data structure. Depending on the information type/level requested, data gets decoded differently.
For ***dwLevel == BS300IF_DIAG_LVL_CTRL*** the structure contains information about a tightening controller. In this case the structure type is *BSIFDiagInfo_Ctrl* and it contains the following elements:

Element		Description	
dwSize		Total size of this structure (in bytes)	
dwLevel		Information level of this structure (must be BS300IF_DIAG_LVL_CTRL)	
dwState		Diagnostics data state (one value from the enumeration eBSIFDiagState, see bs300if_misc.h):	
		ebsdisErr	error requesting diagnostics information
		ebsdisUnknown	unknown or no data
		ebsdisDataOK	diagnostics data is valid
		ebsdisOldData	diagnostics data is valid, but not up to date (cached/buffered)
		ebsdisMissing	component not available or not existing
		ebsdisInvSensor	invalid sensor type (MC1/MC2)
The following fields are only valid, if *dwState* is either ebsdisDataOK or ebsdisOldData!			
	dwSerNo	Controller unique serial number	
	dwCtrlType	Controller type (one of the following values)	
		0x0000	SE3xx tightening controller
		0x0012	SEH3xx tightening controller
		0x0080	SE3xx tightening controller (special customized version)
		0x0092	SEH3xx tightening controller (special customized version)
		0x0040	Compact system CS351
	dwToolType	Tool type. This is a combination of LT and Motor type, where the motor type is encoded into the lower 4 bits. To detect an ErgoSpin, check if the lower 4 bits are equal to 0x000A.	
	szName	Controller name (ASCII-Z), e. g. “SE312”	
	szTTNR	Order number (ASCII-Z)	
	szFW_Ver	Firmware version (ASCII-Z), e. g. “1.400”	
	szFW_Build	Firmware build/service pack level (ASCII-Z), e. g. “SP7”	
	szHW_Ver	Hardware version (ASCII-Z)	

For ***dwLevel == BS300IF_DIAG_LVL_TOOL*** the structure contains information about a tightening tool or a tightening spindle (e. g. its primary transducer). In this case the structure type is *BSIFDiagInfo_Tool* and it contains the following elements:

Element	Description
dwSize	Total size of this structure (in bytes)
dwLevel	Information level of this structure (must be BS300IF_DIAG_LVL_TOOL)

dwState	Diagnostics data state (one value from the enumeration eBSIFDiagState, see bs300if_misc.h):	
	ebsdisErr	error requesting diagnostics information
	ebsdisUnknown	unknown or no data
	ebsdisDataOK	diagnostics data is valid
	ebsdisOldData	diagnostics data is valid, but not up to date (cached/buffered)
	ebsdisMissing	component not available or not existing
	ebsdisInvSensor	invalid sensor type (MC1/MC2)
The following fields are only valid, if *dwState* is either ebsdisDataOK or ebsdisOldData!		
dwSerNo	Controller unique serial number	
dwType	Tool/component type (one of the following values, for customized versions add 0x0080)	
	0x0002	MC sensor
	0x0015	MV sensor
	0x0024	VMC sensor
	0x000E	ErgoSpin
szName	Controller name (ASCII-Z)	
szTTNR	Order number (ASCII-Z)	
szCode	Controller code (ASCII-Z)	
szFW_Ver	Firmware version (ASCII-Z), e. g. “1.400”	
szHW_Ver	Hardware version (ASCII-Z)	
dwCheckInt	Check interval	
dwCycles	Cycles ran	

3.8 TIGHTENING ACTUAL VALUE ACCESS

3.8.1 BSIF_JobAdd_BP_SEND_TBLRES_SE

This function sends an actual value request to transfer tightening result data from a single tightening controller channel (SE/CS) to the PC.

```
DWORD BSIF_JobAdd_BP_SEND_TBLRES_SE (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    DWORD dwFlags
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

dwFlags
[in] Flags defining which values to return. The following values are allowed at the moment:

- 0xFFFFFFF2: return only data from last tightening step
- 0xFFFFFFF1: return data from all steps

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends an actual value request to transfer tightening result data from the tightening controller to the PC. The response to this function is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.
In contrast to most other functions of this API this function requires a callback function to be setup which will receive the tightening result values (see function BSIF_Val_SetCallback, chapter 3.8.3). The BS300IF-DLL will call this callback function each time new tightening result data is received.

3.8.2 BSIF_JobAdd_BP_SEND_TBLRES_KE

This function sends an actual value request to transfer tightening result data from the tightening cells controller (KE) to the PC.

```
DWORD BSIF_JobAdd_BP_SEND_TBLRES_KE (
    DWORD dwHandle,
    int iKEID,
    DWORD dwFlags
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

dwFlags
[in] Flags defining which values to return. The following values are allowed at the moment:

- 0xFFFFFFF2: return only data from last tightening step
- 0xFFFFFFF1: return data from all steps

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function sends an actual value request to transfer tightening result data from the tightening controller to the PC. The response to this function is asynchronous. Therefore after calling this function, the functions BSIF_JobState and/or BSIF_JobDone should be used to wait for completion of the request initiated by this function. The exact sequence to use is described in chapter 2.3.
In contrast to most other functions of this API this function requires a callback function to be setup which will receive the tightening result values (see function BSIF_Val_SetCallback, chapter 3.8.3). The BS300IF-DLL will call this callback function each time new tightening result data is received.

3.8.3 BSIF_Val_SetCallback

This function sets a callback function for use with the functions BSIF_JobAdd_BP_SEND_TBLRES_SE and BSIF_JobAdd_BP_SEND_TBLRES_KE.

```
DWORD BSIF_Val_SetCallback (
    DWORD dwHandle,
    pfnBSIF_Val_Callback pCallbackFn,
    void *pUser
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

pCallbackFn
[in] Pointer to a userdefined callback function of type pfnBSIF_Val_Callback. This function will be called each time new tightening results are received after enabling actual value transmission using BSIF_JobAdd_BP_SEND_TBLRES_SE or BSIF_JobAdd_BP_SEND_TBLRES_KE. If pCallbackFn is set to zero/NULL, callbacks are disabled.

pUser
[in] Void pointer which is stored internally and passed as an argument to the callback function each time the callback function is called.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function registers a callback function for use with BSIF_JobAdd_BP_SEND_TBLRES_SE and BSIF_JobAdd_BP_SEND_TBLRES_KE.

3.8.4 BSIF_Val_GetSEData

This function analyses and decodes am actual value data object provided by the actual value callback and copies data into an easy to use data structure.

```
DWORD BSIF_Val_GetSEData (
    BYTE *pObjBuf,
    int iObjLen,
    BSIFSEActVal *pData
);
```

Parameters

pObjBuf
[in] Pointer to the actual value object as provided by the actual value callback.

iObjLen
[in] Integer-Value with the size of the buffer supplied in pSrcBuf.

pData

[in/out] Pointer to a variable length structure of type BSIFSEActVal. Before calling this function the members dwSize and dwLevel must be initialized (dwSize with the available buffer size in bytes, dwLevel with the expected decode structure information level). After a successful call pData will contain the decoded application table data.

Return Values

A return value of zero indicates success. Else an error code is returned (see chapter 2.4 for details).

Remarks

This function decodes an actual value response object from a tightening channel (SE/CS). The resulting structure contains the decoded information according to the information level (dwLevel).
For Information Level 0 (at the moment the only supported information level), the data structure BSIFSEActVal is used. It has the following elements:

Element	Description
dwSize	Total size of this structure (in Bytes)
dwLevel	Information level (must be zero at the moment): 0 = simple, last step info (data structure type BSIFSEActVal)
Sequence	Channels sequence counter for current result
OkNok	Tightening result state: 1 = OK, 0/-1 = NOK
QC1	Quality code first 32 bits
QC2	Quality code second 32 bits
Program	Tightening program number
Channel	Channel number (contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2)
TTAct	Threshold torque actual value
TTDef	Threshold torque defined/set value
T_Min, T_Max, T_Act, T_Def	Torque actual value, limits and defined value of last step
A_Min, A_Max, A_Act, A_Def	Angle actual value, limits and defined value of last step
Row, Column	Row- and column-number of last step (0…n)

3.9 OBSOLETE FUNCTIONS

3.9.1 BSIF_JobResultData

This function directly accesses a jobs result data buffer (obsolete, please use BSIF_Tools_CopyObjData instead).

```
DWORD BSIF_JobResultData (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    BYTE **ppBuf,
    int *piBufLen,
);
```

Parameters

dwHandle
[in] Handle returned by calling BSIF_Init.

iKEID
[in] Unique ID for identifying the communication channel. Defined when calling BSIF_AddKE.

iSEChn
[in] Channel number of the tightening channel to use. The parameter contains encoded <Rack> and <Slot> numbers (see chapter 2.2.2).

ppBuf
[out] Pointer to the internal job objects data buffer that receives the result data (payload, bs-object) from the tightening controller. This buffer is owned by the BS300IF-DLL and may not be freed or modified by the caller. The buffer is only valid until another API call for the same iKEID and iSEChn is made.

piBufLen
[out] Pointer to an integer variable which (after a successful call) receives the size of the job result data buffer.

Return Values

A return value of zero indicates success (*ppBuf and *piBufLen valid). Else an error code is returned (see chapter 2.4 for details).

Remarks

This function is obsolete and only exists due to compatibility reasons. Use BSIF_Tools_CopyObjData instead!

3.9.2 BSIF_Tools_GetCurveData

This function analyses and decodes a curve object (obsolete, use BSIF_Tools_CopyObjData and BSIF_CrvGetCrvData instead).

```
DWORD BSIF_Tools_GetCrvData (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    BSIFCurveData *pData
);
```

3.9.3 BSIF_Tools_SaveCurve

This function saves a curve object into a standard BS300 compatible curve data file (*.crv). This function is obsolete, use BSIF_Tools_CopyObjData and BSIF_Crv_Save instead.

```
DWORD BSIF_Tools_SaveCurve (
    DWORD dwHandle,
    int iKEID,
    int iSEChn,
    const char *pszFilename
);
```

4 SAMPLES

See sample applications provided with the BS300IF setup.

5 CHANGES

Version 2.100.1.2:

- Added the following functions:
  - BSIF_JobAdd_BP_APPNAMTAB
  - BSIF_JobAdd_BP_DEL_ALL_APP
  - BSIF_JobAdd_BP_READ_SRBAPP
  - BSIF_JobAdd_BP_WRITE_SRBAPP
  - BSIF_JobAdd_BP_LOGONREQ
  - BSIF_JobAdd_BP_SYS_APPNAMTAB
  - BSIF_JobAdd_BP_SYS_PRGNAMTAB
  - BSIF_JobAdd_BP_Diag
  - BSIF_App_GetData
  - BSIF_Val_GetSEData
  - BSIF_Val_SetCallback
  - BSIF_Sys_GetLogonData
  - BSIF_Sys_GetAppNameTbl
  - BSIF_Sys_GetPrgNameTbl
  - BSIF_JobSetTimeout
- Added the following new structures
  - tBSIFTightPos
  - BSIFAppData
  - BSIFSysLogonData
  - BSIFSEActVal
  - BSIFCurveParamsV1
  - tBSIFNamedItem
  - BSIFSysNameTable
  - tBSIFChnLogonData
  - tKEIFChnLogonData
- Added automatic system detection (System 300/310/350)

Todo:

- BMS interface
- Better serial port support (for all objects)
- USB-port support (90.0.0.90)
- Pseudocode sample chapter