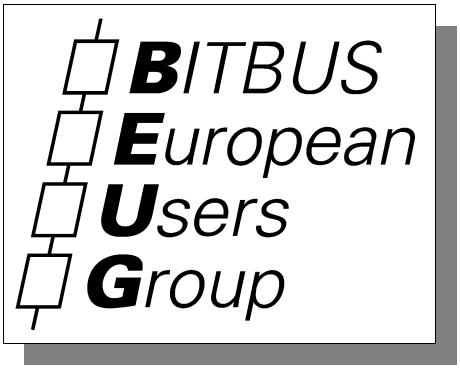
BITBUS
APPLICATION PROGRAMMERS INTERFACE
(BAPI)

A BEUG Recommendation



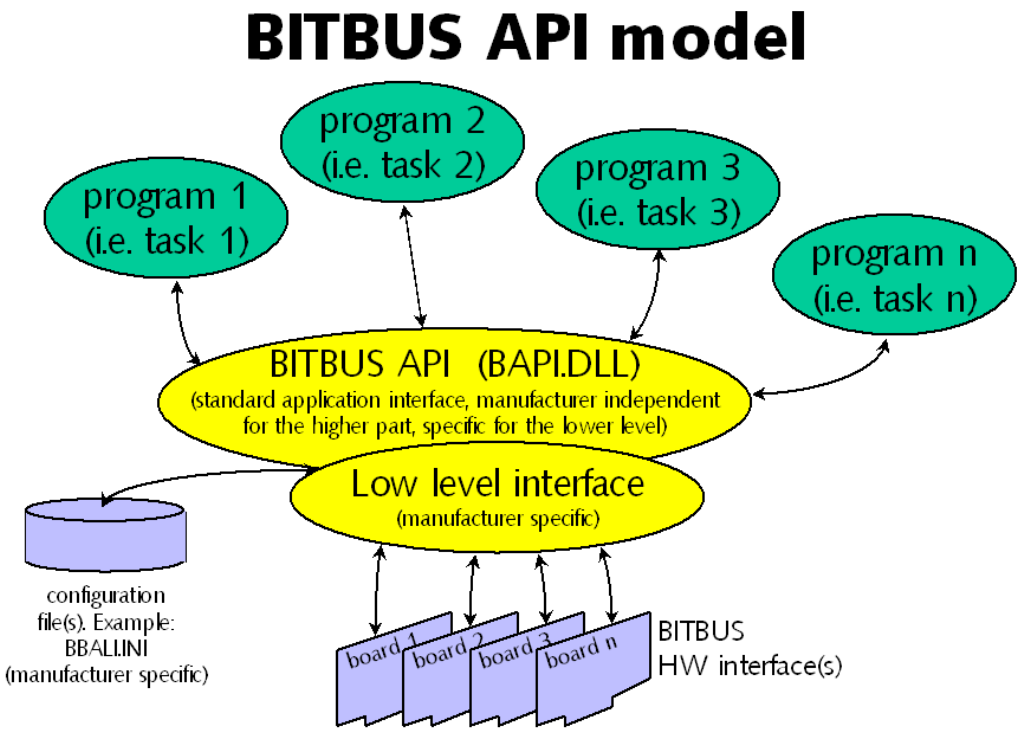
Description:	*BITBUS Application Programmers Interface (BAPI)*
Date:	*May 15, 1998*
Last Change:	*July 21th, 1998, common format for all recommendations, PDF conversion. VG*
	October 15th, 1998, font changed to new std. BEUG font, richer comments
	January 18th, 1999, small changes in GBS command description
Status:	*Accepted by BEUG on May 15, 1998*
Authors:	*Mario Casali, System Electronics S.P.A.* *Bassel Safadi, System Electronics S.P.A.* *Matteo Mondada, CIMSI* *Beggi Oskarsson, Broderson Control Systems A/S* *Volker Goller, mocom software GmbH & Co KG*
Copyright:	*All rights reserved. This document is intellectual property of* *BEUG - The BITBUS European User's Group* *This document can be used for documentation purposes by all BITBUS users and manufactures if their implementation and products meets this specification.*
Contact:	*e-mail: wg1@bitbus.org*

Contents

1. What is it all about?

BAPI is a common BITBUS API (application programmers interface) definition. Suppliers of BITBUS „plug-in“ boards like ISA, PCI or VME Bus boards should support an BAPI library (DLL, LIB) on top of their driver architecture. On top of BAPI higher layer interfaces and tools can be provided in an vendor independent way. Users can replace boards that run out of production by newer boards without redesigning their software - if it uses BAPI only!

Figure 1 shows the BAPI architecture. Please note that BAPI provides a multitasking interface allowing several applications to use BITBUS at a time! Think of diagnostic tools running concurrently with an application!



2. The BAPI reference

2.1. BAPI naming conventions

The BAPI proposal was designed using the C-language. However, naming conventions were designed as flexible as possible to fit into other languages (as Pascal or Java), too.

All public functions start with »Bitbus«. The underline character is not used in function names.

A designer may add private functions to BAPI, however he should not add additional functions starting with »Bitbus«, because the BEUG reserved this prefix for future BAPI extensions. However, if a designer needs additional functions including the »Bitbus« prefix for compatibility needs, he is allowed to do so if he adds a warning text to both the header files and the documentation that these functions are no longer supported and only available for compatibility reasons.

2.2. BAPI constant values

Almost all constant values are present in two version: the RAC and the GBS version.

RAC is »remote access command« and the old style INTEL definition of the base command set every BITBUS controller supports by TASK 0.

GBS is »generic bus services« and the newer IEEE1118 definition of RAC. There are more GBS than RAC commands, RAC is a subset of GBS. RAC definitions are provided for compatibility only and should not be used in newer designs.

2.2.1. The commands

Command code	GBS / BAPI name	Comment
0x000	*GBS_RESET (RAC_RESET)*	*Resets a node. Special care has to be taken because not every implementation will perform a hardware reset!*
0x001	*GBS_CREATE (RAC_CREATE)*	*Create a task. The message carries the tasks ITD (initial task descriptor) address (2 byte, msb first). The address can be extendet using the GBS_EXTEND_ADDR command.*
0x002	*GBS_DELETE (RAC_DELETE)*	*Deletes a running task. The only parameter is the task number returned by GBS_CREATE (or retrieved by GBS_GET_FID)*
0x003	*GBS_GETFID (RAC_GETFID)*	*Retrieves a list of running tasks from a BITBUS node. The message is expected to be an array with the task numbers as an index. The array is filled by the node with the tasks function id codes (FID, a byte). Valid FID codes are:* *0 = NOFID 1 = GBSTASK 0x80..0xfe user FID's and 0xff = NOFID*
0x004	*GBS_PROTECT (RAC_PROTECT)*	*This command allows to lock some commands temporarily. The locking code is a single byte: 0 = UNLOCK, 1 = LOCK, 2 = RDONLY*
0x005	*GBS_READ_IO (RAC_READ_IO)*	*Read specified IO ports. The addresses and the data are expected to be bytes.* *The GBS_EXTEND_ADDR command can be used to enlarge the addressing range to 24 bits (0..0xffffff). However, all byte addresses within a message are extended with the same base address so that all ports accessed by a single message must be within the same 256-byte page.*
0x006	*GBS_WRITE_IO (RAC_WRITE_IO)*	*Write a value to the specified IO ports. The addresses and the data are expected to be bytes. The address range can be extended, see GBS_READ_IO for details.*
0x007	*GBS_UPDATE_IO (RAC_UPDATE_IO)*	*Write a value to the specified IO ports and read them back immediately. The addresses and the data are expected to be bytes. The address range can be extended, see GBS_READ_IO for details.*
0x008	*GBS_UPLOAD_DATA (RAC_UPLOAD_DATA)*	*Read a chunk of memory from a given 16-bit address. The address can be extended to 32-bit using the GBS_EXTEND_ADDR command.*
0x009	*GBS_DOWNLOAD_DATA (RAC_DOWNLOAD_DATA)*	*Write a chunk of memory to a given 16-bit address. The address can be extended to 32-bit using the GBS_EXTEND_ADDR command.*
0x00A	*GBS_OR_IO (RAC_OR_IO)*	*The data in the message is bitwise ored with the current port value and write the result back to the port immediately. The addresses and the data are expected to be bytes. The address range can be extended, see GBS_READ_IO for details.*

Command code	GBS / BAPI name	Comment
0x00B	*GBS_AND_IO* *(RAC_AND_IO)*	*The data in the message is bitwise anded with the current port value and write the result back to the port immediately. The addresses and the data are expected to be bytes. The address range can be extended, see GBS_READ_IO for details.*
0x00C	*GBS_XOR_IO* *(RAC_XOR_IO)*	*The data in the message is bitwise xored with the current port value and write the result back to the port immediately. The addresses and the data are expected to be bytes. The address range can be extended, see GBS_READ_IO for details.*
0x00D	*GBS_WRITE_SCRATCHPAD* *(RAC_WRITE_SCRATCHPAD)*	*Write values to the specified address in scratch pad ram. The addresses and the data are expected to be bytes. This command was/is used to access i8044/i80152 internal RAM for an example. The use is implementation depending.* *The address range can be extended, see GBS_READ_IO for details.*
0x00E	*GBS_READ_SCRATCHPAD* *(RAC_READ_SCRATCHPAD)*	*Read values from the specified address in scratch pad ram. The addresses and the data are expected to be bytes. This command was/is used to access i8044/i80152 internal RAM for an example. The use is implementation depending.* *The address range can be extended, see GBS_READ_IO for details.*
0x00F	*GBS_GET_NODE_INFO* *(RAC_GET_NODE_INFO)*	*This commands replies some node information:* *6 byte of node id ("i8044"), all ASCII.* *2 bytes version of node firmware ("21"), all ASCII* *1 byte (byte #9) supported message length* *Additional vendor and/or implementation depending information can be added behind byte #9.*
0x010	*GBS_OFFLINE* *(RAC_OFFLINE)*	*Reset the communication between a master and a specific node. Used to fix a bug within older i8044 processors ...*
0x011	*GBS_UPLOAD_CODE* *(RAC_UPLOAD_CODE)*	*Read a chunk of memory from a given 16-bit address. If the processor supports different address spaces for code and data memory, the commands GBS_UPLOAD_DATA (0x008) and GBS_UPLOAD_CODE can be used to access both spaces at will.* *The address can be extended to 32-bit using the GBS_EXTEND_ADDR command.*
0x012	*GBS_DOWNLOAD_CODE* *(RAC_DOWNLOAD_CODE)*	*Write a chunk of memory to a given 16-bit address. If the processor supports different address spaces for code and data memory, the command GBS_DOWNLAOD_DATA (0x009) and GBS_DOWNLOAD_CODE can be used to access both spaces at will. If the code memory is an EEPROM or FLASH memory, a suitable programming algorithm is used.* *The address can be extended to 32-bit using the GBS_EXTEND_ADDR command.*

Command code	GBS / BAPI name	Comment
0x013	*GBS_READ_REGISTER*	*This command is new with IEEE1118. Allows to read a processors special function registers (SFR). This command is implementation depended. Use is not recommended, because a wrong programming of an SFR may have dramatic drawbacks.*
0x014	*GBS_WRITE_REGISTER*	*This command is new with IEEE1118. Allows to read a processors special function registers (SFR). This command is implementation depended. Use is not recommended, because a wrong programming of an SFR may have dramatic drawbacks.*
0x015	*GBS_GET_TIME*	*Read the nodes clock information.*
0x016	*GBS_SET_TIME*	*Sets the nodes clock information to a given value.*
0x017	*GBS_SUSPEND_TASK*	*Stop execution of an task. The task number must be provided.*
0x018	*GBS_RESUME_TASK*	*Restarts a previously suspended task. The task number must be provided.*
0x019	*GBS_DEFINE_SERVICE*	*Install a user supplied command (0xc0..0xff).*
0x01A	*GBS_GET_TASK_ID*	*Get a tasks id (task number). The function ID is given as a parameter.*
0x0BF	*GBS_EXTEND_ADDR*	*This command can be used to extend the addressing range of an command. Memory up- and downloads can be boosted up to 32-bit and IO commands up to 24-bit. The GBS_EXTEND_ADDR modifies a given command - so two commands (GBS_EXTEND_ADDR + the original command) are included within a single message. There is no loss in performance using this command!* *The extended address can be a page address on 80x51 systems supporting paging, a 16-Bit segment address on a 80x86 system or a linear address (68000, 386, 80251, ...)*
0x0C0	*GBS_USER_SERVICE_START*	*Defines the first user defined command*
0x0FF	*GBS_USER_SERVICE_END*	*Defines the last user defined command*

2.2.2. Special parameters and constants

Parameter code	*Name*	*RAC*	*GBS*	*Comment*
0x000	*GBS_UNPROTECTED* *(RAC_UNPROTECTED)*	✓	✓	*Enable GBS/RAC access*
0x001	*GBS_RW_PROTECTED* *(RAC_RW_PROTECTED)*	✓	✓	*Disable GBS/RAC access*
0x002	*GBS_WRITE_PROTECTED* *(RAC_WRITE_PROTECTED)*	-	✓	*Set GBS/RAC access to read-only.*

Mask value	Mask name	Related Bitbus header byte	Comment
0x080	MT_FLAG	flags	Message type. If set, the message is a response (a message from slave to master). This field should be handled by BAPI to prevent errors.
0x040	SE_FLAG	flags	Source extension or 5th bit of source task number. Source extension is used to route a BITBUS command message to a host connected to a BITBUS master node (e.g. PC plug-in board). This field should be handled by BAPI to prevent errors.
0x020	DE_FLAG	flags	Destination extension or 5th bit of destination task number. Destination extension is used to route a BITBUS response message to a host connected to a BITBUS slave node
0x010	TR_FLAG	flags	Direction bit for host interface transfers.
0x0F0	SRC_TSK	src_dest	Mask the source task number from the src_dest field. The result must be shifted 4 times right to get a valid task number. This field should be handled by BAPI to prevent errors.
0x00F	DST_TSK	src_dest	Mask the destination task number from the src_dest field.

2.2.3. Error codes

Error code	Name	Related GBS/RAC commands	Comment
0x000	GBS_OK, GBS_ERR_OK	All except GBS_CREATE	All works fine!
0x01..0x1f	Task number, response	GBS_CREATE	The GBS_CREATE returns the task number of the new created task in com_res.
0x20..0x7f	User defined errors	-	Reserved for user defined commands
0x080	GBS_ERR_NO_DEST_TASK	ALL incl. non-GBS tasks	This error can have two sources: If a message is send to a task that does not exist, the link layer will answer the message signalling this error. If a GBS/RAC command uses an task number as an parameter (e.g. GBS_DELETE), this error is returned if the requested task does not exist.
0x081	GBS_ERR_TASK_OVFL	GBS_CREATE	If no more tasks can be created by the OS, this error is replied.

Error code	Name	Related GBS/RAC commands	Comment
0x082	GBS_ERR_REGISTER_OVFL	GBS_CREATE	If no more tasks can be created because of a lack of register sets.
0x083	GBS_ERR_DUPLICATE_FID	GBS_CREATE	If there is already another task running with the same FID in use, this error is returned - no task is created.
0x084	GBS_ERR_NO_BUFFERS	GBS_CREATE	If there is not enough memory to create a stack for the new task, this error is returned - no task is created.
0x085	GBS_ERR_BAD_TASK_PRTY	GBS_SUSPEND, GBS_RESUME, GBS_CREATE	The current priority or the priority found in the ITD is invalid.
0x086	GBS_ERR_BAD_TASK_DESC	GBS_CREATE	If the ITD descriptor found is not valid - no task is created.
0x087	GBS_ERR_NO_MEMORY	All GBS/RAC	An internal operation fails because of lack of memory.
0x088	GBS_ERR_BAD_PROC_ADDR	GBS_DEFINE_SERVICE	The address parameter of the command is not valid.
0x090	GBS_ERR_TIME_OUT	ALL incl. non-GBS tasks	A node can not be reached or a communication link breaks caused by successive timeouts. Note that not all breaks in communication can be signalled and therefor a layer7 timeout has to be maintained by the application.
0x091	GBS_ERR_PROTOCOL	ALL incl. non-GBS tasks	General protocol error (sequencing error, ..)
0x093	GBS_ERR_NO_DEST_DEVICE	ALL incl. non-GBS tasks	Bad destination address (0 or 255)
0x095	GBS_ERR_PROTECTED	ALL incl. non-GBS tasks	The access to this command is protected by a previous GBS_PROTECT command.
0x096	GBS_ERR_UNKNOWN_CMD	ALL incl. non-GBS tasks	The command is not known or not supported.
0x097	GBS_ERR_BAD_CMD_LEN	ALL incl. non-GBS tasks	This error can be used to signal an application that the node expected a longer message.
0x98..0xfd	-	-	Reserved by BEUG for future use
0x0FE	GBS_ERR_BAD_SERVICE		The command is not supported. This error is an IEEE1118 add-on. It can extends the GBS_ERR_UNKNOWN_CMD error code.
0x0ff	GBS_ERR_GENERAL	ALL	Unspecified error. Should not be used if there is a chance to avoid it!

2.2.4. BAPI error codes and miscellaneous constants

Error codes are related to functions returning BBHANDLE or INT32.

Error code	*Name*	*Comment*
0L	*BAPI_OK, BAPI_ERR_OK*	*BAPI function call success code*
-1L	*BAPI_ERR_TIMEOUT*	*If „BitbusWaitMsg“ run into a timeout condition, this code is returned.*
-2L	*BAPI_ERR_NO_BOARD*	*If an open (master or slave) fails because there is no board installed, this is the error returned.*
-3L	*BAPI_ERR_NO_CONNECTION*	*This error is returned by the BAPI function „BitbusGetMsgLength“ if the slave „node“ can not be reached.*
-4L	*BAPI_ERR_RESET_FAIL*	*„BitbusReset“ fails caused by unspecified reasons.*
-5L	*BAPI_ERR_INVALID_TID*	*The task id in the „src_dest“ field of the BITBUS header is not valid. See „3.1. BAPI in a multi-tasking environment“ for more information.*
-6L	*BAPI_ERR_INVALID_FID*	*The FID is not valid*
-7L	*BAPI_ERR_INVALID_HANDLE*	*The bitbus handle returned by an open (master or slave) call is not or no longer valid.*
-8L	*BAPI_ERR_BUFF_TOO_SHORT*	*You try to send a message that will not fit into the given buffer. It is a good practice to retrieve the usable message length by calling „BitbusGetMsgLength“ to prevent this error.*
-9L	*BAPI_ERR_INVALID_FLAGS*	*The flags setting in the „flags“ field of the BITBUS header is illegal. See „3.1. BAPI in a multi-tasking environment“ for more information.*
-100L and lower	*BAPI_ERR_USER*	*Reserved for implementation specific errors. An implementor should avoid to use user errors.*

Value	*Name*	*Comment*
255	*BAPI_MAX_MSG_LEN*	*Maximum message length specified by IEEE1118*
-1L	*BAPI_WAIT_FOREVER*	*Eternal wait condition (BitbusWaitMsg)*
0	*BAPI_LOCAL_SCOPE*	*Parameter for BitbusGetMsgCount*
1	*BAPI_GLOBAL_SCOPE*	*Parameter for BitbusGetMsgCount*

2.3. BAPI Datatypes

BAPI specifies several different data structures. The most important is the BITBUS message itself.

2.3.1. *Data packing*

Manage packing of structures at byte boundary. See sample code for WATCOM C/C++:

```
#if defined(__WATCOM__)     // enable packing
#pragma pack(push)
#pragma pack(1)
#elif defined(__STDC__)
#pragma pack(1)
#endif
// place definition here ...
#if defined(__WATCOM__)     // restore packing
#pragma pack(pop)
#endif
```

2.3.2. Base types

```
typedef unsigned char       UINT8;            // unsigned 8 bit integer
typedef unsigned short      UINT16;           // unsigned 16 bit integer
typedef unsigned long       UINT32;           // unsigned 32 bit integer
typedef unsigned char       BYTE;             // same as UINT8
typedef unsigned short      WORD;             // same as UINT16
typedef unsigned long       LWORD;            // same as UINT32

typedef char                INT8;             // signed 8 bit integer
typedef short               INT16;            // signed 16 bit integer
typedef long                INT32;            // signed 32 bit integer

typedef INT32               BBHANDLE; // returned by BitbusOpen calls
```

2.3.3. BITBUS message

```
typedef struct {
    BYTE  _res1;          /* PRIVATE VALUE                      */
    BYTE  _res2;          /* PRIVATE VALUE                      */
    BYTE  len;            /* 7 + LENGTH OF DATA FIELD           */
    BYTE  flags;          /* ROUTING BITS                       */
    BYTE  node;           /* MASTER: SLAVE ADDRESS.             */
                          /* SLAVE : NEVER TOUCH!               */
    BYTE  src_dest;       /* OWN/TARGET TASK NUMBER (0-15/0-15) */
    BYTE  com_res;        /* REQUEST: GBS-/USER-COMMAND         */
    BYTE  data[248];      /* DATA FIELD                         */
    BYTE  _res3;          /* PRIVATE VALUE                      */
} BitbusMsg;

typedef BitbusMsg *pBitbusMsg;
```

2.3.4. GBS-TASK time & date format

```
typedef struct {
    BYTE zone,              /* TIME ZONE   : 0 = GMT, 8 = PST */
         offset,            /* TIME OFFSET : 0..59 MINUTES    */
         day_of_week,       /* 1..7, MONDAY = 1.              */
         year,              /* 1980 = 0, 2235 = 255.          */
         month,             /* 1..12, JANUARY = 1.            */
         day,               /* 1..31.                         */
         hour,              /* 0..23.                         */
         min,               /* 0..59.                         */
         sec;               /* 0..59.                         */
} GbsTime;

typedef GbsTime *pGbsTime;
```

2.3.5. Implementation dependent types

These types maybe be defined and supplied by the BAPI implementor.

```
typedef struct {
    INT32         dummy;
} BitbusOpenData;

typedef BitbusOpenData *pBitbusOpenData;
```

2.4. BAPI Function reference

For each supported operating system, macro BAPICALL must be defined, macro could be different at build time and at use time.

Future implementations of BAPI.H could add other routines having prefix "Bitbus"; no implementor and user should name his own routines using this prefix. „BAPICALL“ is a plattform depending macro. See 3.2. for sample macros for WATCOM C–Compiler.

Functions not implemented for some reasons should return graceful.

2.4.1. BAPI DLL/LIB naming conventions

BAPI implementations should follow the BAPI naming rules for DLL (dynamic link library) and LIB (library) files. The table below gives the file name that will be extended by the file type code („.DLL“, „.LIB“ ..).

Operating system	***File name***
MSDOS / PCDOS / NWDOS et. al.	BAPIDOS
DOS, 32-BIT protected mode	BAPID32
Microsoft Windows 3.1	BAPIW16
Microsoft Windows 95, 98	BAPIW32
Microsoft Windows NT	BAPINT
IBM OS/2	BAPIOS2
UNIX (LINUX)	BAPIIX
Microware OS/9	BAPIOS9
QNX	BAPIQNX

I. Function: BitbusOpenMaster

Returns a handle to the entire network associated to "BitbusDevice" ("BBUS0", ... "BBUSxx"); application "AppName" will then be enabled to act as a network master, that is to send order messages and receive reply messages.

Parameters:

"AppName" is an individual name used to identify an application.

"BitbusDevice" is the logical name of the the Bitbus card; valid names are: "BBUS0", "BBUS1", ... , "BBUSxx" (being xx a decimal number).

"pData" is a pointer to a data structure containing additional, implementation dependent parameters required to appropriately open the Bitbus card; see definition of type "BitbusOpenData".

Return value:

If greater than or equal to 0, connection has successfully been established

Possible errors:

BAPI_ERR_NO_BOARD if no hardware was found

Prototype:

```
BBHANDLE BAPICALL BitbusOpenMaster   (char           *AppName,
                                      char           *BitbusDevice,
                                      BitbusOpenData *pData);
```

II. Function: BitbusOpenSlave

Returns a handle to the card associated to "BitbusDevice" ("BBUS0", ... "BBUSxx"); application "AppName" will then be enabled to act as a slave of the network, that is to wait for order messages and send reply messages.

Parameters:

"AppName" is an individual name used to identify an application.

"BitbusDevice" is the logical name of the the Bitbus card; valid names are: "BBUS0", "BBUS1", ... , "BBUSxx" (being xx a decimal number).

"TaskId" is a number between 1 and 15, so that API can route incoming order messages to the appropriate application; slave connections can be at most 15; some implementation could let this number range up to 31.

"FunctionId" is a number between 0x002 and 0x0FE; a network master can exchange messages with a slave application, by knowing its FID and by using command GBS_GETFID to retrieve its TID.

"pData" is a pointer to a data structure containing additional, implementation dependent parameters required to appropriately open the Bitbus card; see definition of type "BitbusOpenData".

Return value:

If greater than or equal to 0, connection has successfully been established; if

Possible errors:

BAPI_ERR_NO_BOARD if no hardware was found

BAPI_ERR_INVALID_FID, if the fid is allready in use.

Prototype:

```
BBHANDLE BAPICALL BitbusOpenSlave    (char           *AppName,
                                      char           *BitbusDevice,
                                      BYTE            TaskId,
                                      BYTE            FunctionId,
                                      BitbusOpenData *pData);
```

III. Function: BitbusClose

Close a network connection (either master or slave).

Parameters:

"hdl" is the handle of the connection returned at open time.

Return value:

If equal to 0 (BAPI_OK), connection has successfully been closed

Possible errors:

BAPI_ERR_INVALID_HANDLE if the handle is invalid

Prototype:

```
INT32    BAPICALL BitbusClose        (BBHANDLE        hdl);
```

IV. Function: BitbusSendMsg

Sends an order message (master application) or a reply message (slave application).

Parameters:

"hdl" is the handle of the connection returned at open time.

"pMsg" is a pointer to a message structure which must be allocated and filled by the application program.

Return value:

If equal to 0 (BAPI_OK), connection has successfully been closed

Possible errors:

BAPI_ERR_INVALID_HANDLE if the handle is invalid

BAPI_ERR_BUFFER_TOO_SHORT if the message will not fit into the internal buffers.

*BAPI_ERR_INVALID_FLAGS if the flags are not correct

*BAPI_ERR_INVALID_TID if the source task id is not valid

*BAPI should prevent this types of errors by managing this bits and bytes automatically.

Prototype:

```
INT32    BAPICALL BitbusSendMsg      (BBHANDLE        hdl,
                                      pBitbusMsg      pMsg);
```

V. Function: BitbusWaitMsg

Waits for an order message (slave application) or for a reply message (master application) for a specified amount of time, at most.

Parameters:

"hdl" is the handle of the connection returned at open time.

"pMsg" is a pointer to a message structure which must be allocated by the application program; the structure is filled by the function, if a message has actually been received.

"tout" is the maximum number of milliseconds the function has to wait for the message; value 0 is used for polling; value -1 (BAPI_WAIT_FOREVER) is used for indefinite wait.

Return value:

Length of received message. If "tout" is 0 and returned length is 0, polling has been unsuccessful. If returned length is greater than 0 (7 to 255), see field "com_res" of message structure (might be GBS_ERR_TIMEOUT). If returned length is less than 0, an error condition occurred.

Possible errors:

BAPI_ERR_INVALID_HANDLE if the handle is invalid

BAPI_ERR_TIMEOUT if the function runs into a timeout condition

BAPI_ERR_BUFFER_TOO_SHORT if the message will not fit into the internal buffers.

Prototype:

```
INT32    BAPICALL BitbusWaitMsg      (BBHANDLE        hdl,
                                      pBitbusMsg      pMsg,
                                      INT32           tout);
```

VI. Function: BitbusReset

Used by master applications only, it resets a slave node.

Parameters:

"hdl" is the handle of the connection returned at open time.

"node" is the node to be reset (1 to 250).

Return value:

If equal to 0 (BAPI_OK), reset has been successful

Possible errors:

BAPI_ERR_INVALID_HANDLE if the handle is invalid

BAPI_ERR_RESET_FAIL if the reset function fails

Prototype:

```
INT32    BAPICALL BitbusReset        (BBHANDLE        hdl,
 BYTE            node);
```

VII. Function: BitbusGetMsgLength

Used by master applications only, it returns the maximum length of messages which can be exchanged with a specified slave node.

Parameters:

"hdl" is the handle of the connection returned at open time.

Return value:

If greater than 0 (7 to 255), function has been successful.

Possible errors:

BAPI_ERR_INVALID_HANDLE if the handle is invalid

BAPI_ERR_INVALID_NO_CONNECTION if the node could not be reached.

Prototype:

```
INT32    BAPICALL BitbusGetMsgLength (BBHANDLE        hdl,
                                      BYTE            node);
```

VIII. Function: BitbusGetMsgCnt

It returns the number of Bitbus messages handled by the API for the current connection (BAPI_LOCAL_SCOPE) or for all the connections associated to the same device (BAPI_GLOBAL_SCOPE).

Parameters:

"hdl" is the handle of the connection returned at open time.

"scope" can assume value BAPI_LOCAL_SCOPE or value BAPI_GLOBAL_SCOPE.

Return value:

If greater than or equal to 0, function executes successfully.

Possible errors:

BAPI_ERR_INVALID_HANDLE if the handle is invalid

Prototypes:

```
INT32    BAPICALL BitbusGetMsgCnt    (BBHANDLE   hdl,
                                       WORD       scope);
```

IX. Function: BitbusGetAppNames

It returns a list of application names, for all applications which are currently connected to the same device.

Parameters:

"hdl" is the handle of the connection returned at open time.

"buffer" is the character string which will contain the list of names; each name is terminated by '\n'; the whole list is terminated by '\0'.

"length" is the size of "buffer"; if it is less than the required length, the list of names will be truncated and an error code will be returned.

Return value:

If greater than or equal to 0, function has been successful.

Possible errors:

BAPI_ERR_INVALID_HANDLE if the handle is invalid

BAPI_ERR_BUFFER_TOO_SHORT if the name list will not fit into the users buffer.

Prototype:

```
INT32    BAPICALL BitbusGetAppNames  (BBHANDLE   hdl,
                                      char      *buffer,
                                      WORD       length);
```

3. BAPI implementation notes

3.1. BAPI in a multi-tasking environment

In a multi tasking environment, BAPI implementation should take care that the source / destination task fields and the flags in the BITBUS message header fit with the BAPI channel handle values to prevent message miss routing and to prevent complex error handling.

3.2. Sample OS depending macros for WATCOM-C

```
#undef BAPICALL
#if defined(BUILDING_BAPI)
  #if defined (__OS2__)
    #define BAPICALL EXPENTRY
  #elif defined (__DOS__)
    #define BAPICALL __cdecl
  #elif defined (__MDOS__)
    #define BAPICALL __cdecl
  #elif defined (__WINDOWS__)
    #define BAPICALL __far pascal __export
  #elif defined (__NT__)
    #define BAPICALL __declspec(dllexport) __stdcall
  #elif defined (__WINDOWS_386__)
    #define BAPICALL __declspec(dllexport) __stdcall
  #endif
#else
```

```
  #if defined (__OS2__)
    #define BAPICALL APIENTRY
  #elif defined (__DOS__)
    #define BAPICALL __cdecl
  #elif defined (__MDOS__)
    #define BAPICALL __cdecl
  #elif defined (__WINDOWS__)
    #define BAPICALL __far pascal __export
  #elif defined (__NT__)
    #define BAPICALL __declspec(dllimport) __stdcall
  #elif defined (__WINDOWS_386__)
    #define BAPICALL __declspec(dllimport) __stdcall
  #endif
#endif
```