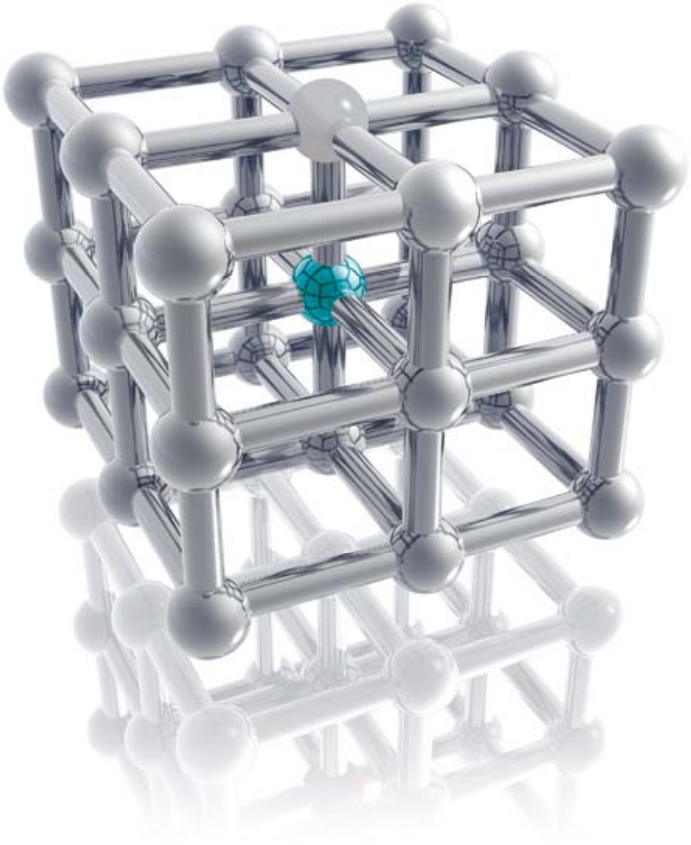

Driver BUFOR – User's Manual

Doc. No. ENP5026
Version: 28-10-2007

ASKOM Sp. z o. o., ul. Józefa Sowińskiego 13, 44-121 Gliwice,
tel. +48 (0) 32 3018100, fax +48 (0) 32 3018101,
http://www.askom.com.pl, e-mail: office@askom.com.pl

1. Bufor

A comprehensive transmission channel named BUFOR was implemented in **asix** visualization system. The channel allows exchanging data between ASMEN communication manager and any process data transmission software developed by the user.

The *BUFOR* channel is realized by two programs:

- a general purpose driver (hereinafter referred to as *BUFOR*, supplied by ASKOM), which ensures communication between ASMEN communication manager and any user program,
- a user driver (hereinafter referred to as *USERDRV*), implemented as a process operating in WINDOWS /NT/2000 environment.

Data exchange between the programs under consideration is realized by a memory mapped file. Synchronization of access to the memory mapped file takes place with the use of a *mutex* object.

2. Description of User Driver Resources

The memory mapped file used to exchange data between *USERDRV* and *BUFOR* must be created by *USERDRV*. Structures of the data that describe process variables contained in this file must be prepared by *USERDRV* on the driver initialization stage and made accessible via *UserDesc* descriptor located in the beginning of the memory mapped file. Below the *UserDesc* descriptor structure is presented:

```
struct UserDesc
{
#define USER_NAME 15
byte name[USER_NAME];          // driver name
word progMajor                 // USERDRV version number
word progMinor;                // USERDRV version minor number
word major;                    // protocol version major number
word minor;                    // protocol version minor number
word flags;                    // operation characteristics
word status;                   // driver status
word items;                    // number of variables handled
                               // (number of variable descriptors in VarDesc
                               // table)
DWORD varDescOffset;           // location of the variable descriptors table
// in memory mapped file (offset with respect to the beginning
// of memory mapped file)
HANDLE synchroMutex;           // handle of mutex synchronization object
word globalRead;               // global read flag
word globalWrite;              // global write flag
word rezerwa1;
word rezerwa2;
};
```

Below the meanings of the *UserDesc* structure fields are presented:

Name - information meaning

progMajor	- information meaning
progMinor	- information meaning
major	- for verification of *BUFOR* and *USERDRV* version conformity
minor	- for verification of *BUFOR* and *USERDRV* version conformity
flags	- defines operating mode of *USERDRV* . The field is a bit flag. The following flags are defined at present: **BFLAG_USERTIME** - *USERDRV* sets read times for **BFLAG_AUTOREFRESH** variables - *USERDRV* automatically refreshes data without waiting for signal from *BUFOR*. The remaining bits of the *flags* field must be cleared.
Status	- current status of *USERDRV*. It has not been defined how the field is to be used yet. It should be cleared at beginning.
globalRead	- the flag is used to transfer read orders from *BUFOR* to *USERDRV*. Detailed description of their usage can be found in read operation description. The field should be initialized to 0,
globalWrite	- the flag is used to transfer write orders from *BUFOR* to *USERDRV*. Detailed description of their usage can be found in write operation description. The field should be initialized for the value of 0,
synchroMutex	- handle of *mutex* synchronization object used to synchronize *BUFOR* and *USERDRV's* access to the common memory mapped file.
rezerwa1	- reserve field, initialized to 0,
rezerwa2	- reserve field, initialized to 0.

The user program name, memory mapped file name and parameters transferred to the user program as command line parameters are declared when the *BUFOR* is being parameterized (section *ASMEN* in 'ini' file).

3. Process Variable Descriptors

Process variables are described by descriptors included in the *VarDesc* table. The *VarDesc* table can be found in the memory mapped file. Position of the table with respect to the beginning of the memory mapped file is defined in *varDescOffset* field in *UserDesc* descriptor.

Any operations performed by the *BUFOR* and *USERDRV* are based on the contents of *VarDesc* descriptors.

Descriptors are initialized partially by *USERDRV* on *USERDRV* process setup (entering the type, number of variables, read and write buffer offsets), whereas their full initialization is completed by *BUFOR* driver during ASMEN process variables' database creation stage.

Below the *VarDesc* descriptor structure is presented:

```
typedef struct VarDesc VARD

struct VarDesc
{
word type;              // type of variable
word items;             // maximum number of variable components (variable can
                        be
```

```
                                // represented by a table)
dword period;                   // intervals (in seconds) between variable value refreshing
                                operations
/* fields used during the read/refreshing operations */
DWORD readBuffer;               // position of the beginning of the read/refreshing operation
                                buffer
                                // with respect to the beginning of the memory mapped file
word readDataStat;              // status of the variable value read out
word readQuery;                 // status of read/refreshing commands
word readResponse;              // status of read/refreshing command realization
dword readTime;                 // time of read operation sent to ASMEN
byte readRez1;                  // reserved field
word readRez2;                  // reserved field
/* fields used during write operations */
DWORD writeBuffer;              // position of the beginning of the read/refreshing operation
                                buffer
                                // with respect to the beginning of the memory mapped file
word writeDataStat;             // status of variable value write operation
word writeQuery;                // status of write commands
word writeResponse;             // status of write command realization
byte writeRez1;                 // reserved field
word writeRez2;                 // reserved field
};
```

USERDRV initializes the following fields of *VARD* structure:

type - used by *BUFOR* to check whether the variable type conforms to the type expected by the calculation function declared for this variable. List of variable types can be found in **bufor.h** header file,

items - maximum number of items (for tabular variables),

readBuffer - position of the buffer, used to read variables, with respect to the beginning of the memory mapped file. The buffer size should be the same as the variable size,

writeBuffer - position of the buffer, used to write variables, with respect to the beginning of the memory mapped file. The buffer size should be the same as the variable size. When it is equal to NULL, the variable write operation is denied,

readRez1 - reserve field. Initialized to 0,

readRez2 - reserve field. Initialized to 0,

writeRez1 - reserve field. Initialized to 0,

writeRez2 - reserve field. Initialized to 0,

The remaining fields are set by *BUFOR* when creating the list of variables. Alternatively, they are synchronizing fields. In any case, the fields should be initially set to 0.

The meanings of the fields initialized by *BUFOR* are as follows:

period - refreshing time (in seconds). Set by *BUFOR*. It is designed for information purpose only. ASMEN counts the refreshing times and will order the read operations at appropriate times. *USERDRV*, which automatically refreshes variables, can use the *period* field to define the intervals between read operations,

readTime - in *readTime* field, *USERDRV* passes the variable read time as a number of seconds since 01.01.1980. When the

BFLAG_USERTIME flag has not been set in *UserDesc* descriptor, the driver has not to handle this field.

readDataStat
readQuery
readResponse - read operation synchronization fields. Meaning described in Read Operation,

writeDataStat
writeQuery
writeResponse - write operation synchronization fields. Meaning described in Write Operation,

4. BUFOR Driver Parameterization

BUFOR parameterization requires that the following information should be given:

- name of the memory mapped file, which is used to exchange data between *BUFOR* and *USERDRV*,
- name of the program to be loaded by *BUFOR* as *USERDRV*,
- command line parameters to be sent to *USERDRV*.

Below the user driver declaration is given. It is performed by USER.EXE, which exchanges data with *BUFOR* driver via PLIK_MMF memory mapped file. Three parameters are passed to USER.EXE.

TEST = BUFOR, PLIK_MMF, USER, PAR1 PAR2 PAR3

whereas *TEST* is the name of the logical channel that uses the user driver.

5. Process Variable Declaration

Symbolic address of a process variable is as follows:

I*index*

where:

index - index of a given variable from the driver (*VarDesc*) variable descriptors' table. Index of the first variable is 1.

The remaining parameters in the process variable declaration have typical meaning.

6. Obtaining Access to Common Data

Due to independent operation of *BUFOR* and *USERDRV* it is necessary to ensure synchronized access to common data during performing the variable read /write operations via common memory mapped file.

Synchronization of that access is performed basing on API WIN32 procedures that operate on *mutex* object. It is assumed that *mutex*, which synchronizes access to memory mapped file will be created by *USERDRV* on its initialization stage and that mutex's handle will be passed to *BUFOR* via *synchroMutex* field in *UserDesc* descriptor.

Below there is an example of a source code to create mutex:

```
HANDLE TestMutex;
struct UserDesc UserDesc;

short CreateSemaphore(void)
{
if (TestMutex = CreateMutex(NULL,   FALSE, NULL)  == NULL)
        {
        /* mutex creation error. Return with error */
        return ERROR;
        }

/*  passing the handle to BUFOR */
up->synchroMutex = TestMutex;
return OK;
}
```

Below there is an example code that performs synchronized access to the memory mapped file:

```
void HandleCommonData(void)
{
        /* waiting for access to memory mapped file */
        WaitForSingleObject(TestMutex,INFINITE);
        /* safe operation on common data */

                .
                .
                .
        /* release of access right to memory mapped file */
        ReleaseMutex(TestMutex);
}
```

7. Read Operation

The following fields of *VARD* structure are involved in read operations: *readBuffer*, *readDataStat*, *readQuery*, *readResponse*, *readTime*. The *readTime*, *readDataStat* fields and *readBuffer* buffer may only be changed by *USERDRV*. The *readResponse* and *readQuery* fields are changed by both parties. Definitions of the flags used in the description and operation execution status are defined in **bufor.h** header file.

The following operations are performed for *USERDRV* that does not refresh of process variable values automatically.

Operation sequence of *BUFOR* driver – initialization of read operation :

1) obtain access right,
2) if *INREAD* bit in *readResponse* field is set (previous read is still performed), read initialization is given up,
3) clear *readResponse* field,
4) set the *REQUEST* flag in *readQuery* field,
5) repeat steps 2/, 3/ and 4/ for all the variables for which read operation is being initialized,
6) set *READ_REQUEST* value in *globalRead* field in *UserDesc* descriptor,
7) release access right.

Operation sequence for *USERDRV* driver – initialization of read operation on *BUFOR* request:

1) check *globalRead* field in *UserDesc* descriptor. If it is different from 0, clear it and then review all the variables according to steps 2/ to 7/,
2) obtain access right,
3) heck *REQUEST* flag in *readQuery* field, If flag is set, perform steps 4/ and 5/,
4) lear *readQuery* field,
5) execute the internal initialization of physical read operation and set *INREAD* flag in *readResponse* field,
6) execute steps 3/, 4/ and 5/ for all variables,
7) release access right.

Operation sequence for *USERDRV* – completion of read operation:

1) obtain access right,
2) enter new value into *readBuffer*, set *readDataStat* and *readTime* fields (if *USERDRV* sets the read time on its own),
3) set *DONE* flag and clear *INREAD* flag in *readResponse* field,
4) repeat steps 2/ and 3/ for all the variables, which read operation has been completed,
5) release access right.

Operation sequence for *BUFOR* – read operation completion:

1) obtain access right,
2) for all the variables, which *DONE* flag is set in *readResponse* field, retrieve the contents of *readBuffer* buffer and *readDataStat* and *readTime* fields,
3) release access right.

8. Write Operation

The following fields are involved in write operations of process variable : *writeBuffer*, *writeDataStat*, *writeQuery*, *writeResponse*. The *writeDataStat* field may be changed by *USERDRV* only. The *writeBuffer* buffer is changed by *BUFOR* only. The *writeResponse* and *writeQuery* fields are changed by both the parties. Definitions of any flags used in the description and status of operation execution are defined in **bufor.h** header file.

Operation sequence for *BUFOR* – initialization of write operation:

1) obtain access right,
2) if *INWRITE* bit in *writeResponse* field is set (previous write operation is still being performed), initialization is given up (error status or retry to perform write operation after some time),

3) clear *writeResponse* field,
4) enter a new value in *writeBuffer*, setting *REQUEST* flag in *writeQuery* field,
5) repeat steps 2/, 3/ and 4/ for all variables written in a specific cycle of B*UFOR* operation,
6) set *WRITE_REQUEST* value in *globalWrite* field in *UserDesc* descriptor,
7) release access right.

Operation sequence for *USERDRV* – write operation initialization:

1) check *globalWrite* field in *UserDesc* descriptor. If it is different from 0, clear it and then review all the variables in a way described below,
2) obtain access right,
3) if *REQUEST* flag is set in *writeQuery* field, perform steps 4/ and 5/,
4) clear *writeQuery* field,
5) perform internal initialization of physical write operation and set *INWRITE* flag in *writeResponse* field,
6) repeat steps 3/, 4/ and 5/ for all the process variables,
7) release access right.

Operation sequence for *USERDRV* – completion of write operation:

1) obtain access right,
2) set *writeDataStat* field,
3) clear *INWRITE* flag and set *DONE* flag in *writeResponse* field,
4) repeat steps 2/ and 3/ for all variables, which write operation has been completed,
5) release access right.

Operation sequence for *BUFOR* – completion of write operation:

1) obtain access right,
2) if *DONE* flag is set in *writeResponse* field, then the status is retrieved from *writeDataStat* field,
3) repeat step 2/ for all variables, which write operation has been initialized,
4) release access right.

9. Bufor.h Header File

Definitions of any flags used in the description and status of operation execution are defined in **bufor.h** header file. Below the content of this file is presented.

```
#define BFLAG_USERTIME      1
#define BFLAG_AUTOREFRESH   2

/* read or write operation request flag */
#define REQUEST             1

/* flags that describe statue of read or write operation  */
#define INREAD              1
#define INWRITE             1
#define DONE                2

/* types of process variables */
```

```
#define  BTYPE          1       // byte
#define  ITYPE          2       // integer
#define  WTYPE          3       // unsigned integer
#define  LTYPE          4       // long
#define  DWTYPE         5       // unsigned long
#define  FPTYPE         6       // float type


/* status of operation completion returned by driver */
#define  AVD_GOOD       0       // o.k.
#define  AVD_BAD        1       // data does not exist
#define  AVD_FAIR       2       // data from previous read operation.
Current
                                // read operation has completed with error
#define AVD_POOR        3       // data indicated by driver as
                                // not valid
#define AVD_ERROR       4       // data indicated by driver as bad
```

Index

BUFOR driver parameterization 6
Bufor.h header file 9
Description of user driver resources 3
Introduction 3
Obtaining access to common data 6
Process variable declaration 6
Process variable descriptors 4
Read operation 7
Write operation 8

1. BUFOR 3
2. DESCRIPTION OF USER DRIVER RESOURCES 3
3. PROCESS VARIABLE DESCRIPTORS 4
4. BUFOR DRIVER PARAMETERIZATION 6
5. PROCESS VARIABLE DECLARATION 6
6. OBTAINING ACCESS TO COMMON DATA 6
7. READ OPERATION 7
8. WRITE OPERATION 8
9. BUFOR.H HEADER FILE 9